Geodefi Audit

Swap contract

May 2022

By CoinFabrik

Introduction

CoinFabrik was asked to audit some contracts for the Geodefi project. First we will provide a summary of our discoveries, and then we will show the details of our findings.

Scope

The audited files are from the git repository located at https://github.com/Geodefi/Portal-Avax.git. The audit is based on the commit `9c8f28fae9161590ec183b5fa840062138ea7bfc`.

We reviewed the fixes and enhancements on commit `c80c97c76b3994640f2b8c791fe2dd3cdeaa1ce5`.
The audited files are:

- `contracts/WithdrawalPool/Swap.sol`: It contains the `Swap` contract, which implements a swap pool inspired by the StableSwap paper[1], exchanging AVAX and gAVAX.
- `contracts/interfaces/ISwap.sol`: It contains the expected interface for the `Swap` contract.
- `contracts/interfaces/IgAVAX.sol`: It contains the interface definition for the gAVAX tokens. The gAVAX tokens are ERC1155 tokens.
- `contracts/WithdrawalPool/utils/SwapUtils.sol`: Library used in the `Swap` contract. It contains most of the logic to implement the calculations required for the stable swap.
- `contracts/WithdrawalPool/utils/AmplificationUtils.sol`: Library used in the `Swap` contract. It contains the logic to slowly move the A parameter, which is defined in the StableSwap paper.
- `contracts/WithdrawalPool/utils/MathUtils.sol`: Library with some math utilities used in the other files.
- `contracts/WithdrawalPool/helpers/OwnerPausableUpgradeable.sol`: Contains the `OwnerPausableUpgradable` contract. Contracts derived from this contract can be paused and resumed.
- `contracts/WithdrawalPool/LPToken.sol`: This token is an ERC20 detailed token with added capability to be minted by the owner. It is used to represent user's shares when providing liquidity to swap contracts.

[1] https://curve.fi/files/stableswap-paper.pdf

The scope of the audit is limited to those files. No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Analyses

Without being limited to them, the audit process included the following analyses:

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

Summary of Findings

We found no critical or medium issues. Several minor issues were found. Also, several enhancements were proposed.

Three of the minor issues were acknowledged and one resolved. Also one of the proposed enhancements was made.

Security Issues

ID	Title	Severity	Status
MI-01	Solidity Compiler Version	Minor	Acknowledged
MI-02	Excessive Trust in Swap Contract	Minor	Acknowledged
MI-03	Inconsistent Exchange Rate	Minor	Acknowledged
MI-04	Missing Feature	Minor	Resolved

Security Issues Found

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit can have four distinct statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk

Critical Severity Issues

No issues found

Medium Severity Issues

No issues found.

Minor Severity Issues

MI-01 Solidity Compiler Version

All the audited files use the pragma `solidity =0.8.7;` statement. This implies that an old solidity version is being used which may lead into hitting already fixed bugs.

Recommendation

It is better to lock to a specific compiler version (for example, `pragma solidity 0.8.13;`) and keep it up to date, fully testing on each upgrade.

Status

Acknowledged. The development team decided to keep version 0.8.7 given the effort to test all the code in the new version.

MI-02 Excessive Trust in Swap Contract

Location:

- `contracts/WithdrawalPool/Swap.sol`

The gAVAX tokens implement the ERC1155 standard, which by itself does not allow granular handling of transfer rights. An address either can or cannot transfer AVAX on behalf of another address for all the indexes and any available amount. The gAVAX tokens lack functionality similar to the ERC20's `approve()` function. The only way a user can use the `Swap.swap()` and `Swap.addLiquidity()` functions is to relinquish controls of its funds by calling the `IgAVAX.setApprovalForAll()` function.

This lack of functionality, combined with the API defined by the `Swap` contract forces a user to trust that the contract will not take more funds than it should from the same index or from any other index. While the current contract does not seem to be behaving incorrectly, given that the contract is upgradable a non-trusting user should check the balance of its gAVAX for all the indexes where it has funds after calling `Swap.swap()` or `Swap.addLiquidity()` and revoke the approval in the same transaction. This requirement makes it impossible to invoke those functions with an EOA without taking this risk.

It must be noted that the upstream code manages ERC20 tokens, so this is not an issue for them because the tokens have the `approve()` functionality, allowing the user to restrict the amount of tokens to be used by the `Swap.swap()` and `Swap.addLiquidity()` functions.

Recommendation

Add to the gAVAX tokens functionality similar to the ERC20's `approve()` function restricting the amount of tokens that can be transferred by a non-owner of the tokens on behalf of the token owner for any index.

A partial solution may be implementing the `setApprovalForScope()` function defined in EIP-1761[2], which is proposed in EIP-1155, but it would be incomplete because it restricts transfers corresponding to other ids but does not restrict the number of tokens transferred for the same id.

An alternative solution may be to require the user to transfer the tokens to the `Swap` contract prior to the invocation of `Swap.addLiquidity()` or `Swap.swap()` defining a similar API to the one used by Uniswap V2[3]. This solution would effectively rule out the invocation of these functions by an EOA.

Status

Acknowledged.

MI-03 Inconsistent Exchange Rate

Location:

- `contracts/WithdrawalPool/Swap.sol`
- `contracts/WithdrawalPool/SwapUtils.sol`

If a user adds liquidity to the `Swap` contract with AVAX via the `Swap.addLiquidity()` function, obtaining LPTokens, and then redeems those LPTokens via the `Swap.removeLiquidityOneToken()` function to obtain gAVAX, the user will obtain more gAVAX tokens than the tokens obtained from calling the `Swap.swap()` function when swap fees are not 0.

Recommendation

Make sure to make the exchange rate the same, without taking into account the gas consumption. An option is to implement the `Swap.swap()` function as adding liquidity to the pool via the `Swap.addLiquidity()` function, obtaining the liquidity tokens and redeeming them via the `Swap.removeLiquidityOneToken()` function.

Status

Acknowledged. The development team informs us that the difference is minimal and it is the expected behavior in the StableSwap algorithm.

---

[2] https://eips.ethereum.org/EIPS/eip-1761

[3] https://docs.uniswap.org/protocol/V2/reference/smart-contracts/router-02#swaptokensforexacttokens

MI-04 Missing Feature

Location:

- `contracts/WithdrawalPool/Swap.sol:364,388,415`

On the documentation of the 3 functions used to remove liquidity in the `Swap` contract (`removeLiquidity()`, `removeLiquidityOneToken()` and `removeLiquidityImbalance()`) it claims that "Withdraw fee that decays linearly over period of 4 weeks since last deposit will apply". But nowhere in the audited code this decay is implemented.

Recommendation

Remove this phrase from the documentation.

Status

Resolved. The documentation does not claim to have a fee decay anymore. Checked in commit `c80c97c76b3994640f2b8c791fe2dd3cdeaa1ce5`.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

Table

ID	Title	Status
EN-01	Declare Interface Implementation in Swap Contract	Implemented
EN-02	Documentation Not Up to Date in Swap Contract	Not implemented
EN-03	Simplification Opportunities	Not implemented

Details

EN-01 Declare Interface Implementation in Swap Contract

Location:

- `contracts/WithdrawalPool/Swap.sol:30-33`

The `Swap` contract should implement the `ISwap` interface (defined in `contracts/interfaces/ISwap.sol`), as this interface is intended to represent the contract, but it is not declared to implement it in the declaration of the contract.

Recommendation

Add `ISwap` to the `is` clause of the contract and adjust the contract, the interface and its usages.

Status

Implemented. Checked in commit `c80c97c76b3994640f2b8c791fe2dd3cdeaa1ce5`.

EN-02 Documentation Not Up to Date in Swap Contract

Location:

- `contracts/WithdrawalPool/Swap.sol:13-29`

The documentation describing the functionality of the contract reflects the behavior of upstream code and lacks the changes in functionality added by this project.

Recommendation

Change the doc-string to reflect those changes.

Status

Not Implemented. The development team acknowledges this issue and will make a future version of the project upgrading the documentation.

EN-03 Simplification Opportunities

Location:

- `contracts/WithdrawalPool/SwapUtils.sol`

Unlike upstream, the current implementation only handles 2 tokens, AVAX and a single gAVAX (identified by its index). This fact allows several code simplifications that would also allow some gas savings. For example, see code near lines 341, 386 and 450 of `contracts/WithdrawalPool/SwapUtils.sol`.

Recommendation

Do not generalize to n different tokens in the `SwapUtils` code.

Status

Not Implemented. The development team did not implement these changes. They informed us that the Solidity compiler handles this.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest for other

stakeholders of the project, including users of the audited contracts, owners or project investors.

Upstream

Most of the audited code was based on the code in the git repository at https://github.com/saddle-finance/saddle-contract/, which in turn is based on the original StableSwap implementation at https://github.com/curvefi/curve-contract/. Upstream code was audited several times and those audits are available at https://github.com/saddle-finance/saddle-audits.

We used commit `b47e81db2d9456b81519b1105b7256ff99dd96a1` as a base to compare upstream code with the audited code. We found the following differences in the `Swap` contract:

- Upgraded solidity version to 0.8.4 and removed the safe-math library because solidity 0.8.4 has the required functionality by default.
- Upstream supports the exchange between any number of ERC20 tokens, the audited code only handles the exchange between AVAX and gAVAX. Only one index of the gAVAX coin is supported on each instantiation of the `Swap` contract.
- Upstream does not support changing the equilibrium price based on an oracle.
- The `Swap.getDebt()` function is not available in the upstream code. This function is used in the gAVAX staking process, which is outside of the scope of this audit.

Differences with StableSwap Paper

The Swap contract is based on the StableSwap paper published in https://curve.fi/files/stableswap-paper.pdf. The StableSwap paper assumes that the equilibrium price in the exchange of the coins and the leverage are constants. Both of them can change in this implementation.

The changes in the A parameter were already present in the upstream code. The logic is in the `contracts/WithdrawalPool/utils/AmplificationUtils.sol` file. and it changes slowly. The parameter changes slowly (it cannot double or halven in less than 14 days). The change can only be triggered by the contract owner.

The change of the equilibrium price was added in this implementation. This is implemented in the new `SwapUtils._priced*()` functions located in lines 125-165 of `contracts/WithdrawalPool/utils/SwapUtils.sol`. The

development team informed us that the price changes given by the oracle are constrained by the code in `contracts/Portal/utils/StakeUtilsLib.sol`, which is outside the scope of this audit. In particular see the `sanityCheck()` function in line 295. This code ensures that the price will increase monotonically and that it will not increase more than 0.2% in 30 minutes.

Gains or losses generated by the equilibrium price change or the leverage change will be absorbed by the liquidity providers.

Upgrades

The owner of all the analyzed contracts (`OwnerPausableUpgradable`, `Swap` and `LPToken`) can upgrade the code of the contract with any other code.

The development team informed us that neither Swap nor LPToken are effectively upgradeable because they are Clones created by Portal.

Centralization

Besides the upgrades, the owner of the `Swap` contract can:

- change the fee percentages (both swap and admin). The swap fees are capped at 1% and the admin fees are a part of those.
- slowly change the `A` parameter, changing the leverage of the exchange. The rate of change is restricted by the contract.
- pause and resume the swap operation. A user interacting with a paused `Swap` contract cannot use it to exchange AVAX and gAVAX, but if the user has LPTokens it can exchange them back to avax and gavax at the current exchange rate, removing the liquidity.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

OwnerPausableUpgradeable

Given that `OwnerPausableUpgradeable` is an abstract contract, contracts derived from it may have other roles and or other capabilities for the owner role.

Owner

The owner of the contract can:

- Pause and resume contracts derived from this contract.
- Replace the code of the contract with new logic, while keeping all the assets.

Swap

It has all the roles defined for the `OwnerPausableUpgradeable` contract. No new roles were added.

Owner

Besides all the abilities defined in the parent contracts, the owner can:

- Withdraw the admin fees.
- Set the admin fee.
- Set the swap fee.
- Initiate the change of the A parameter[4].
- Stop the change of the A parameter immediately.

LPToken

Owner

The owner of this contract can:

- Replace the code of the contract with new logic.
- Mint new tokens and assign them to any address.

Changelog

- 2022-05-20 – Initial report based on commit `9c8f28fae9161590ec183b5fa840062138ea7bfc`.
- 2022-06-10 – Fixes reviewed on commit `c80c97c76b3994640f2b8c791fe2dd3cdeaa1ce5`.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the Geodefi project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.

[4] see the StableSwap paper for details on this parameter