Geodefi Audit

Portal and gAVAX

August 2022

By CoinFabrik

Introduction **4**
Scope 4
Analyses 5
Summary of Findings **5**
Security Issues 5
Security Issues Found **6**
Severity Classification 6
Issues Status 6
Critical Severity Issues 6
Medium Severity Issues 7
Minor Severity Issues 7
MI-01 The Senate May Approve a Proposal Multiple Times 7
MI-02 Phantom Overflows When Multiply and Divide 7
MI-03 Missing Zero Check 8
MI-04 Retracted 8
MI-05 Excessive Capability For Withdrawal Pool 8
MI-06 Excessive Trust in Withdrawal Pool 9
Enhancements **9**
Table 9
Details 10
EN-01 gAVAX Contract Roles Simplification 10
EN-02 Proposal Mechanism Improvements 10
EN-03 Governance-Change Mechanism 11
EN-04 Gas Optimization Accessing Datastore 11
Other Considerations **12**

Centralization 12

gAVAX 12

Portal 13

Upgrades 13

gAVAX 13

Portal 13

Modified OpenZeppelin ERC1155 support 13

Unavailable Operations When Portal is Paused 14

Actions on Proposal Approval 15

Oracle Interactions and Price Setting in Portal 15

gAVAX-Portal Interaction 16

Privileged Roles **16**

gAVAX 16

Approved Operator 16

Minter 16

Pauser 17

Oracle 17

Interface 17

Default admin 18

Portal 18

Governance 18

Oracle 18

Senate 18

Maintainer 19

Controller 19

Changelog **20**

Introduction

CoinFabrik was asked to audit the 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 `895e79900b757f0f29edc105eb0c9bb8be1dde1c`.

Fixes were checked on commit `8f0397777b86812e9528483e21e8091334a9d4f8`.

The audited files are:

- `contracts/Portal/gAVAX.sol`: gAVAX token implementation.
- `contracts/Portal/helpers/ERC1155SupplyMinterPauser.sol`: Base contract for ERC1155. Most of the code has been taken from the OpenZeppelin library. See Modified OpenZeppelin ERC1155 Support in Other Considerations for details.
- `contracts/Portal/Portal.sol`: It contains the `Portal` contract, which is a decentralized Minter that builds a trustless staking Ecosystem for any service provider.
- `contracts/Portal/utils/StakeUtilsLib.sol`: It contains the logic for staking gAVAX tokens.
- `contracts/Portal/utils/GeodeUtilsLib.sol`: It contains functions responsible for administration of Geode Portal, including functions related to "limited upgradability" with Senate & proposals.
- `contracts/Portal/utils/DataStoreLib.sol`: Storage management tool designed to create a safe and scalable storage layout with the help of ids and keys.

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. In particular, it must be noted that the `contracts/WithdrawalPool/LPToken.sol` file was audited in the Geodefi's audit made in May 2022 by us.
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.

Two minor issues were resolved, three minor issues were acknowledged and one minor issue was retracted as we found it was not an issue in the first place.

Security Issues

ID	Title	Severity	Status
MI-01	The Senate May Approve a Proposal Multiple Times	Minor	Resolved
MI-02	Phantom Overflows When Multiply and Divide	Minor	Acknowledged
MI-03	Missing Zero Check	Minor	Resolved
MI-04	Retracted	Minor	Retracted
MI-05	Excessive Capability For Withdrawal Pool	Minor	Acknowledged

ID	Title	Severity	Status
MI-06	Excessive Trust in Withdrawal Pool	Minor	Acknowledged

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.
- **Retracted**: After further analysis, we found that the issue originally reported was not an actual issue.

Critical Severity Issues

No issues found.

Medium Severity Issues

No issues found.

Minor Severity Issues

MI-01 The Senate May Approve a Proposal Multiple Times

Location:

- `contracts/Portal/utils/GeodeUtilsLib.sol:256-285`

The senate may approve several times a proposal in the same block if Portal.`approveProposal()` is invoked in different transactions of the same block. This may lead to inconsistencies such as orphan withdrawal pools.

The problem is that the deadline of the proposal is checked in line 261

```
require(
   self._proposalForId[id].deadline >= block.timestamp,
   "GeodeUtils: proposal expired"
);
```

And the deadline is set in line 281 as the current timestamp

```
self._proposalForId[id].deadline = block.timestamp;
```

So the check would pass again if run in the same block.

Recommendation

Check that the proposal deadline is strictly bigger than the current timestamp (>).

Status

Resolved.

MI-02 Phantom Overflows When Multiply and Divide

Location:

- `contracts/Portal/utils/StakeUtilsLib.sol: 306,352,362`

When multiplying and dividing 3 256-bit numbers (`a*b/c`), the intermediate result of multiplying `a*b` may exceed 256 bits even if the final result is under 256bits. This will trigger an overflow, reverting the transaction.

Recommendation

In order to avoid the unwanted overflow, implement the multiply and divide operation to handle bigger numbers after the multiplication. If multiplying 256bit numbers, it should handle 512-bit results.

Status

Acknowledged. The development team says that "Because the parameters are limited with small numbers, such as `periodsSinceUpdate` can only be 1-2-... , we think solving this issue is unnecessary and increases code complexity."

MI-03 Missing Zero Check

Location:

- `contracts/Portal/Portal.sol: 473`

The `Portal.setDefaultInterface()` function lacks a zero check for the address of the `_newDefault interface`, but this check is implemented in the `Portal.initialize()` function (see line 121 of `contracts/Portal/Portal.sol`).

Recommendation

Add the zero check.

Status

Resolved according to the recommendation.

MI-04 Retracted

Retracted. After further analysis, we found that the MI-04 issue originally reported was not an actual issue.

MI-05 Excessive Capability For Withdrawal Pool

Location:

- `contracts/Portal/utils/StakeUtilsLib.sol:772`

The planet's withdrawal pool is given the power to operate on behalf of the `Portal` contract in the gAVAX contract for all the ids in line 772 of `contracts/Portal/utils/StakeUtilsLib.sol`, even though it needs to operate only in tokens associated with the withdrawal pool's id:

```
getgAVAX(self).setApprovalForAll(WithdrawalPool, true);
```

Recommendation

Add a function in gAVAX to approve a single id and use that to give the capability to the `WithdrawalPool`.

Status
Acknowledged.

MI-06 Excessive Trust in Withdrawal Pool

Location:

- `contracts/Portal/utils/StakeUtilsLib.sol: 226-232`

While doing a buyback, the library calls the `swap()` function of the corresponding withdrawal pool and, instead of measuring the obtained amount of gAVAX it trusts the value provided by the withdrawal pool, which may or may not be the actual amount obtained. An incorrect amount may lead to burning or transferring an incorrect amount of gAVAX.

Recommendation
Instead of trusting that the amount of gAVAX reported is correct, ask for the balance before and after doing the swap and check the minimum amount obtained after that.

Status
Acknowledged. According to the development team: "Since Withdrawal Pool contract is not upgradable, we believe it is okay to trust".

Enhancements

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

Table

ID	Title	Status
EN-01	gAVAX Contract Roles Simplification	Not implemented
EN-02	Proposal Mechanism Improvements	Not implemented
EN-03	Governance-Change Mechanism	Not implemented
EN-04	Gas Optimization Accessing Datastore	Implemented

Details

EN-01 gAVAX Contract Roles Simplification

Location:

- `contracts/Portal/gAVAX.sol`
- `contracts/Portal/helpers/ERC1155SupplyMinterPauser.sol`

As described in lines 135-136 of `contracts/Portal/gAVAX.sol`, the oracle, pauser and minter roles are intended to be given to a single address, the `Portal` address. So there is no need to separate those roles nor have support for this role to be had by several addresses simultaneously.

Recommendation

So we recommend to not use `AccessControl`-type roles, and instead to derive from OpenZepellin's `Ownable` contract, giving the owner all the privileges of those roles. This would significantly simplify the `gAVAX` contract reducing the attack surface.

Status

Not implemented.

EN-02 Proposal Mechanism Improvements

Location:

- `contracts/Portal/Portal.sol`
- `contracts/Portal/utils/GeodeUtilsLib.sol`

The approval proposal mechanism in the `Portal` contract has some drawbacks. In particular we found that:

1. There are no mechanisms to withdraw a proposal before expiration if not approved.
2. There are no mechanisms to remove planets or operators. If for any reason a misbehaved planet or operator must be removed, an emergency code upgrade will be required.
3. As mentioned in the Actions on Proposal Approval subsection of the Other Considerations section, proposals with unknown types will be registered as approved but no other action will be taken.

Recommendation

In order to solve these problems we recommend to:

1. Add a mechanism for the governance to withdraw proposals that were not approved yet.
2. Add additional proposal types to remove operators and planets (or some other similar mechanism).
3. Do not allow unknown proposal types to be proposed (and approved).
4. Do not allow ids already taken as planets or operators to be used in proposals for new planets and operators.

Status

Not implemented.

EN-03 Governance-Change Mechanism

Location:

- `contracts/Portal/Portal.sol`

In the Portal contract the governance cannot be changed without a code update.

Recommendation

Add a mechanism to change the governance to a new address. Take into account that:

1. All the withdrawal-pools ownerships should be transferred to the new governance (see line 672 of `Portal.sol`) or the ownership should be set to the `Portal` contract itself and the governance should interact with the `Portal` contract in order to exercise the ownership of the withdrawal pool.
2. Given the risk of a bogus address set as the governance. The new address should accept the role before the old address relinquishes this capability.
3. We don't have an opinion regarding whether the change of governance should be senate approved or not.

Status

Not implemented.

EN-04 Gas Optimization Accessing Datastore

Location:

- `contracts/Portal/utils/StakeUtilsLib.sol: 515,521,522,525,526`
- `contracts/Portal/Portal.sol: 774,775,778,779`

When reading from the Datastore, if the same value has to be used several times in the same function it is cheaper to store the value in a local variable and use that value.

For example see the repetition when fetching `unclaimedFees` in lines 520-526 of `contracts/Portal/utils/StakeUtilsLib.sol` (repetition underlined).

```
if (
  _DATASTORE.readUintForId(_poolId, "unclaimedFees") >
  _DATASTORE.readUintForId(_poolId, "surplus")
) {
  // the difference between unclaimedFees and the surplus is the debt for the fees.
  uint256 debtInFees = _DATASTORE.readUintForId(_poolId, "unclaimedFees") -
    _DATASTORE.readUintForId(_poolId, "surplus");
```

Recommendation

Look for functions that use the same value stored in a function and refactor them to put the value in a local variable and use that in the rest of the function.

Status

Implemented.

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.

Centralization

In both `gAVAX` and `Portal` contracts, efforts were made to make them less centralized.

gAVAX

In the `gAVAX` contract it must be noted that the minter role is very powerful, as it can gain the oracle and pauser role. It also can mint new tokens for any address (and any id) and set the interface for any id.

The interface for a given id may burn tokens of that id for any account, via the `burn()` function. This makes the minter addresses capable of minting and burning tokens for any id.

It must also be noted that a default admin can grant or revoke the minter, pauser, oracle and default admin role and by default this role is given to the deployer of the contract.

Portal

In the Portal contract it must be noted that initially both the governance role and the senate role are taken by the _GOVERNANCE address passed via the initialize() function, effectively making the contract centralized in practice before a different senate is set via the proposal mechanism given that the upgrade mechanism can deploy arbitrary contract code and is proposed by the governance and approved by the senate.

It must also be noted that the governance cannot be changed to any other address and will by the owner of all the withdrawal pools handled by the Portal (see lines 671-672 of contract/Portal/Portal.sol, function approveProposal()).

```
address WithdrawalPool = STAKEPOOL.deployWithdrawalPool(DATASTORE, _id);
Ownable(WithdrawalPool).transferOwnership(GEODE.GOVERNANCE);
```

Upgrades

gAVAX

The gAVAX contract has no mechanism for upgrading it.

Portal

In order to upgrade the Portal contract, the following procedure must be followed:

1. The governance must make an upgrade proposal via the newProposal() function. The new implementation must be passed in the _CONTROLLER parameter.
2. The senate then must approve the proposal via the approveProposal() function.
3. After that upgradeTo() can be called by anyone, passing the new implementation address. See UUPSUpgradeable documentation in OpenZeppelin for more information.

Modified OpenZeppelin ERC1155 support

The utilities in the OpenZeppelin/contracts library for support of the ERC1155 standard were changed to support the development of the gAVAX token. The modified libraries reside in the contracts/Portal/helpers/ERC1155SupplyMinterPauser.sol file. The code was taken from the OpenZeppelin git repository at https://github.com/OpenZeppelin/openzeppelin-contracts/ and based on commit `cb3f2ab900e39c5ab6e0de6663edf06f573b834f`. The following changes were made:

- The `_doSafeTransferAcceptanceCheck()` method of the `ERC1155` contract visibility was changed from private to internal in order to allow it to be overridden in child contracts.
- The `ERC1155PresetMinterPauser` was renamed as `ERC1155SupplyMinterPauser`.
- The `ERC1155SupplyMinterPauser` gained a new base contract, `ERC1155Supply`.
- The `ERC1155SupplyMinterPauser._beforeTokenTransfer()` function was marked as overriding the `ERC1155Supply._beforeTokenTransfer()` aswell.

It must be noted that the `_doSafeTransferAcceptanceCheck()` function is overridden in the `gAVAX` contract, allowing the interface of the token id to transfer (or mint) tokens to contracts that do not implement the `IERC1155Receiver` interface. This functionality is not available for batch transfers (or batch mints).

Unavailable Operations When Portal is Paused

The governance can pause the `Portal` contract. While paused, the following operations cannot be made:

- Change the controller associated with an id via the `changeIdCONTROLLER()` function.
- Change the maintainer associated with an id via the `changeIdMaintainer()` function.
- Set the default interface via the `setDefaultInterface()` function.
- Active and deactivate the operator associated with an id via the `activateOperator()` and `deactivateOperator()` functions.
- Set the interface associated with a planet via the `setPlanetInterface()` function.
- Set the PBank associated with an operator and a planet via the `setPBank()` function.
- Obtain the PBank associated with an operator and a planet via the `getPBank()` function.[1]
- Add new proposals via the `newProposal()` function.
- Approve proposals via the `approveProposal()` function.
- Approve a new senate via the `approveSenate()` function.

[1] The `getPBank()` function was not intended to run only when not paused. This restriction was lifted for commit `8f0397777b86812e9528483e21e8091334a9d4f8`.

  - Set new prices and increase balances for operators via the `reportOracle()` function.
  - Pay debt associated with a staking pool via the `payDebt()` function.
  - Claim a surplus associated with a staking pool via the `claimSurplus()` function.
  - Claim fees associated with a planet via the `claimFee()` function.

Actions on Proposal Approval

In the `Portal` contract, unless the proposal is a senate proposal, when a proposal is approved by the senate via the `approveProposal()` function, the controller in the proposal is set as the controller for the id of the proposal. After that, other actions are made based on the type of the proposal. Hereunder is a list of proposal types and actions:

- Type 2 (upgrade): The controller associated with the proposal is approved for upgrade.
- Type 4 (operator): The controller in the proposal is set as `"maintainer"` for the proposal id.
- Type 5 (planet): Several actions are made:
  - The controller in the proposal is set as `"maintainer"` for the proposal id.
  - A new interface is cloned from the `DEFAULT_INTERFACE` and associated with the proposal id
  - A new withdrawal pool is deployed, associated with the proposal id and its ownership is transferred to the governance.
- Other types: No extra actions.

New senate proposals (type 1) have a different workflow. When 2/3 of the planets vote for a new senate via the `approveSenate()` function, the controller of the new senate proposal will be instituted as the new senate.

Oracle Interactions and Price Setting in Portal

In the `Portal` contract, the oracle informs increases in the balance for each operator. When this happens the logic located in the `reportOracle()` function defined in the `contracts/Portal/utils/StakeUtilsLib.sol` file is executed, calculating a new price.

In particular, the new price needs to pass the checks in the `_sanityCheck()` function defined in the same file. Here it checks that the price is monotonically ascending and that it does not grow too fast. If any of the checks fail then the transaction informing the new balances for each operator for the given `_poolId` fails. If for any reason those balances must be set in a way that voids this test, the only possible actions are to either upgrade the `Portal` code to change these limitations or report a different balance increase for the operators.

After the price is calculated it is informed to the gAVAX contract via the `setPricePerShare()` function.

gAVAX-Portal Interaction

According to the documentation in lines 135-156 of `contracts/Portal/gAVAX.sol`, the `Portal` contract has the minter, pauser and oracle roles for the gAVAX contract.

On the `Portal` contract side, we assume that the address passed in the `_gAVAX` parameter points to a gAVAX contract. That's why we assume that the contract is trusted.

Privileged Roles

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

gAVAX

Approved Operator

An approved operator may burn or transfer away tokens for any id corresponding to the account that approved the operator.

Approved operators may be added or removed via the `setApprovalForAll()` function.

Minter

An address with the minter role can:

- mint tokens for any id and add them on any account.
- set the interface for each id.
- acquire the pauser and oracle role via the `updateMinterPauserOracle()` function.

- give any other address the minter, pauser, and oracle roles while renouncing these roles via the `updateMinterPauserOracle()` function.

This role is handled using the OpenZeppelin's `AccessControl` base contract. The initial minter is the deployer of the contract (the constructor `msgSender()`).

The development team informed us that this role is given to the `Portal` contract after deployment.

Pauser

An address with the pauser role can pause and unpause the `gAVAX` contract. While the contract is paused all mint, burn and transfer operations will be reverted, but the rest of the functionality will still work. For example, the oracle will be able to set a new price.

This role is handled using the OpenZeppelin's `AccessControl` base contract. The initial pauser is the deployer of the contract (the constructor `msgSender()`).

The development team informed us that this role is given to the `Portal` contract after deployment.

Oracle

An oracle may set the price per share for any id.

This role is handled using the OpenZeppelin's `AccessControl` base contract. The initial oracle is the deployer of the contract (the constructor `msgSender()`).

The development team informed us that this role is given to the `Portal` contract after deployment.

Interface

The interface for an id may transfer away or burn tokens of any account for this id, via the `safeTransferFrom()` and `burn()` functions. It may even transfer tokens to a contract that does not implement the `onERC1155Received()` function or to a contract that would reject the transfer via the same function, not respecting the specification of ERC1155.

The interface-id association may be set and unset by a minter via the `setInterface()` function.

Default admin

All `AccessControl` derived contracts have a default admin role that can add and remove addresses to any role if another admin is not set for that role via the `grantRole()` function. The initial default admin is the deployer of the contract (the constructor `msgSender()`). Given that no admin roles are set in the code, this address can grant and remove the minter, pauser, oracle and default admin roles to any address.

Portal

Governance

The governance can:

- Pause and resume (unpause) the `Portal` via the `pause()` and `unpause()` functions. See Unavailable Operations When Portal is Paused in the Other Considerations section for details.
- Set the operation fee via the `setOperationFee()` function.
- Set the maximum maintainer fee via the `setMaxMaintainerFee()` function.
- Set the default interface via the `setDefaultInterface()` function.
- Make new proposals via the `newProposal()` function.

The address of the governance is passed as a parameter in the `initialize()` function, and cannot be changed without updating the code of the `Portal` contract.

Oracle

The oracle can:

- Increase balances for operators via the `reportOracle()` function for any of the liquidity pools if the `block.timestamp` is in the oracle-active time window. This increase will change the price of the asset according to the calculations in the `reportOracle()` function defined in the `contracts/Portal/utils/StakeUtilsLib.sol` file at line 381.

Senate

The senate can:

- set the maximum operation fee via the `setMaxOperationFee()` function.

- approve all proposals except for type 1 (new senate). See Actions on Proposal Approval in the Other Considerations section for details.

The initial senate is set with the same address as the governance in the `initialize()` function. The senate can work for up to 730 days[2] after the initialization of the contract for the first senate and approval of the new senate for the next ones, as the senate can be changed with a proposal at any time. The governance can propose a new senate, which will be set if 2/3 of the voters chose to approve it.

Maintainer

The maintainer associated with a given id can:

- set and unset the interface associated with the id via the `setPlanetInterface()` function.
- set the PBank associated with the operator and a planet via the `setPBank()` function.
- set the maintainer fee associated with the id via the `setMaintainerFee()` function.
- activate and deactivate an operator for the given id via the `activateOperator()` and `deactivateOperator()` functions.
- pay the debt of the staking pool associated with a pool via the `payDebt()` function.
- claim the surplus of the staking pool associated with a pool via the `claimSurplus()` function.
- pause and unpause staking on the pool associated with the given id via the `pauseStakingForPool()` and `unpauseStakingForPool()` functions.

The maintainer associated with an id is set when the proposal for an operator (type 4) or a planet (type 5) is approved. The initial address is set to the controller of the id. The controller of the address can set a new maintainer using the `changeIdMaintainer()` function.

Controller

The controller associated with a given id can:

- set a new controller for the given id via the `changeIdCONTROLLER()` function.
    - May set an invalid address, making the id effectively have no controller.

[2] Almost 2 years.

- set a new maintainer for the given id via the `changeIdMaintainer()` function.
    - May set an invalid address, making the id effectively have no maintainer.
- vote to approve a new senate.
    - One vote per id will be counted.
    - Valid votes are for ids that correspond to a planet.
    - A new senate will be set if 2/3 of the possible voters choose to approve the new senate.

The controller associated with an id is set when a non-senate proposal is approved.

Changelog

- 2022-08-05 – Initial report based on commit `895e79900b757f0f29edc105eb0c9bb8be1dde1c`.
- 2022-08-23 – Fixes checked on commit `8f0397777b86812e9528483e21e8091334a9d4f8`.
- 2022-08-23 – Minor changes in the Other Considerations and Privileged roles sections, based on feedback from the development team.

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.