shieldify

Geode Finance

SECURITY REVIEW

Date: 18 July 2023

CONTENTS

1. About Shieldify

We are Shieldify Security – a company on a mission to make web3 protocols more secure, cost-efficient and user-friendly. Our team boasts extensive experience in the web3 space as both smart contract auditors and developers that have worked on top 100 blockchain projects with multi-million dollars in market capitalization.

Book an audit and learn more about us at shieldify.org or @ShieldifySec

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Geode Finance

Geode Finance is a novel protocol that offers a staking solution for DAOs to facilitate ETH2 staking to their users. It archives that by enabling any organization to be able to create their own, branded public or private staking pool, removing large development and R&D costs for DAOs and organizations, enabling them to run their own staking as a service offering, quickly and easily, and most importantly in a fully trustless manner. The protocol utilizes modular architecture, making things safer for stakers, and easier for pool providers.

The Most Crucial Component - the Portal

The Portal is the beating heart of the Geode protocol. Here are some of its most important features and functionalities:

- Creation and maintenance of the configurable staking pools.
- Minting new tokens.
- Securing the Ether until it is staked in a validator.
- Onboarding new Operators to the marketplace.
- Management and regulation of the Operator marketplace.
- Allowing new functionalities to be implemented with ease.
- Securing its own codebase from Governance.
- Various tasks of Oracle.

These aims are achieved by:

- Isolated Storage Implementation
- Dual Governance
- Limited Upgradability

3.1. Observations

- Geode Finance cannot upgrade the source code of its contract infrastructure without the approval of its users. Limited Upgradability is used within the Portal, Withdrawal Contract and Liquidity Pool.
- Pool Maintainers can not steal pool fees or pool funds.

- Using Geode Finance's liquidity pool allow your stakers to move their funds between different staking derivatives in just one transaction, with minimal slippage.
- Geode Finance doesn't collect any admin fees on their liquidity pools.
- Geode charges 0% staking-as-a-service fee, and can not change this until March 2025 without the approval of Portal's Senate.
- Pool Owners can charge up to a 10% fee of the yield for maintenance of the pool.
- For the process of changing the pool's owner there are the following warning checks:
    1. "Double check the address of your new Controller."
    2. "If your Pool's owner is not the withdrawal pool's Owner, it will go into `Recovery Mode` until you change its ownership:"
    3. "Changing your controller is easy, however, it will override the ability of the previous controller immediately."
- When the pool's fee is changed, it takes 3 days for the new fee to take effect. Within this 3-day period, the fee cannot be changed again.
- At any given point, a staking pool can have 1 maintainer at most.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired or even gas optimization techniques

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible incentivize
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Audit Summary

The audit duration lasted three and a half weeks and a total of 72 person days have been spent by the three auditors – @ShieldifyMartin, @ShieldifyAnon and @ShieldifyGhost. The project has undergone several audits (both external and internal) prior to this one. It is important to acknowledge that there is still some code that needs to be implemented. Overall, we would like to congratulate the Geode team for the amazing feat that they have accomplished – creating and maintaining such a large and complex codebase with only two developers! With some minor exceptions, the code is very well-documented and written. The test coverage is also comprehensive.

Last but not least, we would like to emphasize that the Geode team has been very communicative and provided detailed answers to all of our questions.

5.1 Protocol Summary

Project Name	**Geode Finance**
Repository	Portal-Eth
Type of Project	Decentralized & Liquid Staking Pools
Audit Timeline	25 days - June 8th - July 2nd
Review Commit Hash	e626ed341a723095c6d22fbfc84081cf7b999e1b
Fixes Review Commit Hash	N/A

5.2 Scope

The following smart contracts were in the scope of the audit:

File	nSLOC
contracts/Portal/gETH.sol	110
contracts/Portal/middlewares/ERC20PermitMiddleware.sol	52
contracts/Portal/middlewares/ERC20Middleware.sol	120
contracts/Portal/modules/DataStoreModule/libs/DataStoreModuleLib.sol	94
contracts/Portal/modules/DataStoreModule/DataStoreModule.sol	40
contracts/Portal/modules/GeodeModule/libs/GeodeModuleLib.sol	111
contracts/Portal/modules/GeodeModule/GeodeModule.sol	89
contracts/Portal/modules/StakeModule/libs/DepositContractLib.sol	64
contracts/Portal/modules/StakeModule/libs/StakeModuleLib.sol	612
contracts/Portal/modules/StakeModule/libs/OracleExtensionLib.sol	139
contracts/Portal/modules/StakeModule/StakeModule.sol	184
contracts/Portal/modules/LiquidityModule/libs/LiquidityModuleLib.sol	507
contracts/Portal/modules/LiquidityModule/libs/AmplificationLib.sol	57
contracts/Portal/modules/LiquidityModule/LiquidityModule.sol	149
contracts/Portal/packages/LiquidityPool.sol	111
contracts/Portal/packages/WithdrawalContract.sol	70
contracts/Portal/Portal.sol	89
Total	**2598**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- Critical and High issues: 0
- Medium issues: 0
- Low issues: 5
- Informational issues: 17
- Gas Optimization issues: 9

ID	Title	Severity
[L-01]	Wrong Storage Gap Value May Break Storage Layout in New Contract Version	Low
[L-02]	Essential Roles Setter Functions Implement Single-Step Role Transfer	Low
[L-03]	Protocol will not Work on Most of the Supported Blockchains due to hardcoded `DEPOSIT_CONTRACT` Address	Low
[L-04]	Usage of `abi.encodePacked` Instead of `abi.encode`	Low
[L-05]	Missing Zero Address Check for `changeSenate` Function	Low
[I-01]	The `deposit` Function Expecting ETH Deposits can check `msg.value` for Sanity and Optimization	Informational
[I-02]	Use `calldata` Instead of `memory` for Function Arguments that do not get Mutated	Informational
[I-03]	The `nonReentrant` Modifier should Occur Before all other Modifiers	Informational
[I-04]	Create a Modifier Only if it will be Used in More than One Place	Informational
[I-05]	Move the Duplicate Checks into a Modifier	Informational
[I-06]	Unused Imports Affect Readability	Informational
[I-07]	Missing Error Messages in `require` and `revert` Statements	Informational
[I-08]	Hardcoded Timestamp Value Should be a Constant	Informational
[I-09]	Change Function Visibility from `public` to `external`	Informational
[I-10]	Use `1e18` Instead of `10**18`	Informational
[I-11]	Use `require` Instead of `assert`	Informational
[I-12]	Use a More Recent Solidity Version	Informational
[I-13]	Function Ordering does not Follow the Solidity Style Guide	Informational
[I-14]	Update External Dependency to the Latest Version	Informational
[I-15]	Missing/Incomplete NatSpec Comments	Informational
[I-16]	Open TODOs	Informational
[I-17]	Typos in Require Statement and Contract Comments	Informational
[G-01]	Using **»1** Instead of **/2** Can Save Gas	Gas Optimization
[G-02]	Using Booleans for Storage Incurs Overhead	Gas Optimization
[G-03]	Use Assembly to Check for `address(0)`	Gas Optimization
[G-04]	Splitting `require()` Statements that Use '&&' Saves Gas	Gas Optimization
[G-05]	No Need to Initialize Variables with Default Values	Gas Optimization
[G-06]	Use custom errors Instead of `require()` with Revert Strings	Gas Optimization
[G-07]	Expressions for `constant` Values Such as a Call to `keccak256()`, Should Use `immutable` Rather than `constant`	Gas Optimization
[G-08]	Array Length Read in Each Iteration of the Loop Wastes Gas	Gas Optimization
[G-09]	Replace Constant Variables `public` Visibility with `private` or `internal`	Gas Optimization

7.Findings

[L-01] Wrong Storage Gap Value May Break Storage Layout in New Contract Version

Severity

Low Risk

Description

`LiquidityPool` and `WithdrawalContract` implements storage gap, which prevents storage collisions in new versions. However, they are meant to be **50 storage-slots-reserved** by the standard. Only complying with it saves from storage collisions, otherwise, they are still possible after the upgrade.

1. In the case of `LiquidityPool` contract, there are 3 `slots` reserved for addressing internal immutables, and the `__gap` is 47 elements long.
2. In the case of `WithdrawalContract` contract, there are 2 `slots` reserved for addressing internal immutables, and the `__gap` is 48 elements long.

Location of Affected Code

File: contracts/Portal/packages/LiquidityPool.sol#L276

```
/**
* @notice keep the total number of variables at 50
*/
uint256[47] private __gap;
```

File: contracts/Portal/packages/WithdrawalContract.sol#L143

```
/**
* @notice keep the total number of variables at 50
*/
uint256[48] private __gap;
```

Recommendation

This calculation is wrong in both contracts and the slot will be 0 and will be equal to 50.

The reason for this is that the compiler doesn't reserve a storage slot for **constant** / **immutable** variables, rather than copying the value of that variable everywhere in the code where that particular value is used.

It's also recommended to use OpenZeppelin's Upgrade Plugin.

Team Response

Acknowledged, will be mitigated.

[L-02] Essential Roles Setter Functions Implement Single-Step Role Transfer

Severity

Low Risk

Description

It's possible to lose these roles under specific circumstances. Because of human error, it's possible to set a new invalid address. When you want to change the address it's better to propose a new one, and then accept the ownership with the new wallet.

Location of Affected Code

File: contracts/Portal/gETH.sol

```
function transferUriSetterRole(
  address newUriSetter
) external virtual override onlyRole(URI_SETTER_ROLE) {

function transferPauserRole(address newPauser) external virtual override
   onlyRole(PAUSER_ROLE) {

function transferMinterRole(address newMinter) external virtual override
   onlyRole(MINTER_ROLE) {

function transferOracleRole(address newOracle) external virtual override
   onlyRole(ORACLE_ROLE) {

function transferMiddlewareManagerRole(
  address newMiddlewareManager
) external virtual override onlyRole(MIDDLEWARE_MANAGER_ROLE) {
```

File: contracts/Portal/modules/GeodeModule/libs/GeodeModuleLib.sol

```
function changeSenate(DualGovernance storage self, address _newSenate)
   external onlySenate(self) {

function changeIdCONTROLLER(
  DSML.IsolatedStorage storage DATASTORE,
  uint256 id,
  address newCONTROLLER
) external onlyController(DATASTORE, id) {
```

Recommendation

Consider using a 2-step process, `approve` and `claim` in two different transactions, instead of a single-step approach. Implement a timelock for important set actions if necessary. Additionally, `Whitelist` and `LPToken` contracts should use `Ownable2Step` and `Ownable2StepUpgradeable` instead of `Ownable` and `OwnableUpgradeable`.

Team Response

A meaningful concern. However, we are keen to keep this structure, as gETH roles will be owned by the smart contracts, which makes it hard to use with 2 step transfers.

Acknowledged, will not be mitigated.

[L-03] Protocol will not Work on Most of the Supported Blockchains due to hardcoded `DEPOSIT_CONTRACT` Address

Severity

Low Risk

Description

This vulnerability has the potential to affect smart contracts that depend on hardcoded addresses for external contracts, particularly in multi-chain deployments. In situations where the addresses of the referenced contracts are altered or the deployment occurs on a different chain, the contracts may encounter difficulties in interacting with the intended contracts and routers, resulting in erroneous behavior and potential malfunctions.

The protocol will not work on most of the supported blockchains due to the hardcoded `DEPOSIT_CONTRACT` address.

Location of Affected Code

File: contracts/Portal/modules/StakeModule/libs/DepositContractLib.sol#L27

```
IDepositContract internal constant DEPOSIT_CONTRACT = IDepositContract(0
    xff50ed3d0ec03aC01D4C79aAd74928BFF48a7b2b);
```

Recommendation

To mitigate this vulnerability, it is advisable to pass the `DEPOSIT_CONTRACT` as constructor parameters when deploying the contract instead of relying on fixed addresses. By allowing the addresses to be configured during deployment, the smart contracts can be utilized across various networks and effectively accommodate changes in contract addresses.

Team Response

We agree that this is an issue that can cause some problems. However, we are keen to use different repositories for different chains as most of the PoS implementations are unique and when the chain changes, many other parts also need to change. Acknowledged, will not be mitigated.

[L-04] Usage of `abi.encodePacked` Instead of `abi.encode`

Severity

Low Risk

Description

The `generateID` function takes a bytes variable with dynamic size, together with a uint256 variable. These arguments are abi encoded and hashed together to produce a unique hash. However, packing differently-sized arguments may produce collisions.

The Solidity documentation states that packing dynamic types will produce collisions, but this is also the case if packing bytes, (which is a shorthand for byte[]) and uint256.

Location of Affected Code

File: contracts/Portal/modules/DataStoreModule/libs/DataStoreModuleLib.sol#L68

```
id = uint256(keccak256(abi.encodePacked(_name, _type)));
```

Recommendation

Unless there's a specific use case to use `abi.encodePacked`, you should always use `abi.encode`. You might need a few more bytes in the transaction data, but it prevents collisions. Additionally, `abi.encode()` pads items to 32 bytes, which will prevent hash collisions (e.g. `abi.encodePacked(0x123, 0x456)=> 0x123456 => abi.encodePacked(0x1, 0x23456)`, but `abi.encode(0x123,0x456)0x0...1230...456`).

Team Response

Acknowledged, will be mitigated.

[L-05] Missing Zero Address Check for `changeSenate` Function

Severity

Low Risk

Description

Contract GeodeModule is missing address validation for the setter function - `changeSenate()`. It is possible to configure the address(0), which may cause issues during execution.

For instance, if address(0) is passed to `changeSenate()` function, it will not be possible to change this address in the future.

Location of Affected Code

File: contracts/Portal/modules/GeodeModule/GeodeModule.sol#L241-L243

```
function changeSenate(address _newSenate) external virtual override {
  GEODE.changeSenate(_newSenate);
}
```

Recommendation

Add a zero-address check on `_newSenate` parameter of `changeSenate()`.

Team Response

Acknowledged, will be mitigated.

[I-01] The `deposit` Function Expecting ETH Deposits can check `msg.value` for Sanity and Optimization

Severity

Informational

Description

The function that expects ETH deposits in their typical flows can check for non-zero values of `msg.value` for sanity and optimization.

Location of Affected Code

File: contracts/Portal/modules/StakeModule/StakeModule.sol#L398-L419

```
function deposit(
  uint256 poolId,
  uint256 price,
  bytes32[] calldata priceProof,
  uint256 mingETH,
  uint256 deadline,
  address receiver
)
  external
  payable
  virtual
  override
  whenNotPaused
  nonReentrant
  returns (uint256 boughtgETH, uint256 mintedgETH)
{
  if (!STAKE.isPriceValid(poolId)) {
    STAKE.priceSync(DATASTORE, poolId, price, priceProof);
  }

  (boughtgETH, mintedgETH) = STAKE.deposit(DATASTORE, poolId, mingETH,
     deadline, receiver);
}
```

Recommendation

It is recommended to add `msg.value` check.

```
++ if (msg.value == 0) revert("StakeModule: msg.value must be non-zero!")
   ;
```

Team Response

Acknowledged, will be implemented.

[I-02] Use `calldata` Instead of `memory` for Function Arguments that do not get Mutated

Severity

Informational

Description

Mark data types as `calldata` instead of `memory` where possible. This makes it so that the data is not automatically loaded into `memory`. If the data passed into the function does not need to be changed (like updating values in an array), it can be passed in as `calldata`. The one exception to this is if the argument must later be passed into another function that takes an argument that specifies `memory` storage.

Location of Affected Code

File: contracts/Portal/helpers/ERC1155PausableBurnableSupply.sol#L646;

```
  function setURI(string memory newuri) public override onlyRole(
     URI_SETTER_ROLE)
```

Recommendation

It is recommended to mark the data type as calldata instead of `memory`.

Team Response

Acknowledged, will be implemented.

[I-03] The `nonReentrant` Modifier should Occur Before all other Modifiers

Severity

Informational

Description

This is a best practice to protect against re-entrancy in other modifiers. It can additionally reduce gas costs if this modifier occurs before all others.

If a function has multiple modifiers they are executed in the order specified. If checks or logic of modifiers depend on other modifiers this has to be considered in their ordering. Some functions have multiple modifiers with one of them being `nonReentrant` which prevents reentrancy on the functions. This should ideally be the first one to prevent even the execution of other modifiers in case of reentrancies.

Location of Affected Code

File: contracts/Portal/Portal.sol#L213

```
function pushUpgrade(
  uint256 packageType
) external virtual override whenNotPaused nonReentrant returns (uint256
   id)
```

File: contracts/Portal/modules/StakeModule/StakeModule.sol#L398

```
function deposit(
  uint256 poolId,
  uint256 price,
  bytes32[] calldata priceProof,
  uint256 mingETH,
  uint256 deadline,
  address receiver
)
  external
  payable
  virtual
  override
  whenNotPaused
  nonReentrant
  returns (uint256 boughtgETH, uint256 mintedgETH)
```

File: contracts/Portal/modules/StakeModule/StakeModule.sol#185

```
function initiateOperator(
  uint256 id,
  uint256 fee,
  uint256 validatorPeriod,
  address maintainer
) external payable virtual override whenNotPaused nonReentrant
```

File: contracts/Portal/modules/StakeModule/StakeModule.sol#185

```
function increaseWalletBalance(
  uint256 id
) external payable virtual override whenNotPaused nonReentrant returns (
   bool)
```

Recommendation

Reorder the modifiers so that `nonReentrant` is first in line.

Team Response

Very good improvement will be implemented.

[I-04] Create a Modifier Only if it will be Used in More than One Place

Severity

Informational

Description

There is no need to create a separate modifier unless it will be used in more than one place. If this is not the case, simply add the modifier code to the function instead.

Location of Affected Code

File: contracts/Portal/modules/GeodeModule/libs/GeodeModuleLib.sol#L301

```
modifier onlyController(DSML.IsolatedStorage storageDATASTORE, uint256 id
    ) {
  require(msg.sender == DATASTORE.readAddress(id, rks.CONTROLLER), "GML:
     CONTROLLER role needed");
  _;
}
```

File: contracts/Portal/packages/WithdrawalContract.sol#L107

```
modifier onlyOwner() {
  require(msg.sender == GEODE.SENATE, "LPP:sender NOT owner");
  _;
}
```

Recommendation

Add the modifier logic into the function directly.

Team Response

Acknowledged, will not be implemented.

[I-05] Move the Duplicate Checks into a Modifier

Severity

Informational

Description

In both `safeTransferFrom` and `burn` functions there is a check that the caller is the token owner or approved address. These checks can be extracted as a modifier that expects address as an input parameter. Extracting checks into a modifier and reusing it in Solidity can provide several benefits, including code reusability, readability, and easier maintenance.

Location of Affected Code

File: contracts/Portal/gETH.sol#L378-L383

```
function safeTransferFrom(
  address from,
  address to,
  uint256 id,
  uint256 amount,
  bytes memory data
) public virtual override {
  require(
    (from == _msgSender()) ||
      (isApprovedForAll(from, _msgSender())) ||
      (isMiddleware(_msgSender(), id) && !isAvoider(from, id)),
    "ERC1155: caller is not token owner or approved"
  );
  _safeTransferFrom(from, to, id, amount, data);
}
```

File: contracts/Portal/gETH.sol#L394-L399

```
function burn(address account, uint256 id, uint256 value) public virtual
    override {
  require(
    (account == _msgSender()) ||
      (isApprovedForAll(account, _msgSender())) ||
      (isMiddleware(_msgSender(), id) && !isAvoider(account, id)),
    "ERC1155: caller is not token owner or approved"
  );

  _burn(account, id, value);
}
```

Recommendation

Extract the checks in a separate modifier.

Team Response

Very good suggestion, however, we are not keen to modify OpenZeppelin's code more than necessary.

[I-06] Unused Imports Affect Readability

Severity

Informational

Description

There are a few unused imports on the codebase. These imports should be cleaned up from the code if they have no purpose.

Location of Affected Code

File: contracts/Portal/packages/LiquidityPool.sol

```
// import {IgETH} from "../interfaces/IgETH.sol";
import {IGeodePackage} from "../interfaces/packages/IGeodePackage.sol";
import {DataStoreModuleLib as DSML} from "../modules/DataStoreModule/libs
   /DataStoreModuleLib.sol";
```

File: contracts/Portal/packages/WithdrawalContract.sol#L12-L14

```
import {DataStoreModuleLib as DSML} from "../modules/DataStoreModule/libs
   /DataStoreModuleLib.sol";
import {GeodeModuleLib as GML} from "../modules/GeodeModule/libs/
   GeodeModuleLib.sol";
import {WithdrawalModuleLib as WML} from "../modules/WithdrawalModule/
   libs/WithdrawalModuleLib.sol";
```

File: contracts/Portal/modules/GeodeModule/libs/GeodeModuleLib.sol#L5

```
import {PERCENTAGE_DENOMINATOR} from "../../../globals/macros.sol";
```

File: contracts/Portal/modules/GeodeModule/GeodeModule.sol#L5

```
import {ID_TYPE} from "../../globals/id_type.sol";
```

Recommendation

Remove the unused imports.

Team Response

Good call, will be fixed.

[I-07] Missing Error Messages in `require` and `revert` Statements

Severity

Informational

Description

When encountering transaction failures or unexpected behavior, the utilization of informative error messages is beneficial for troubleshooting exceptional conditions. Otherwise, inadequate error messages can lead to confusion and unnecessary delays during exploits or emergency situations.

Location of Affected Code

File: contracts/Portal/modules/StakeModule/libs/StakeModuleLib.sol#L456

```
require(versionId > 0);
```

File: contracts/Portal/modules/StakeModule/libs/OracleExtensionLib.sol#L414-L415

```
require(poolIds.length == prices.length);
require(poolIds.length == priceProofs.length);
```

File: contracts/Portal/modules/StakeModule/StakeModule.sol#L112-L113

```
require(_gETH != address(0));
require(_oracle_position != address(0));
```

File: contracts/Portal/modules/LiquidityModule/libs/LiquidityModuleLib.sol#L717

```
revert();
```

Recommendation

Consider adding a descriptive reason in an error string/custom error.

Team Response

Good call, will be fixed.

[I-08] Hardcoded Timestamp Value Should be a Constant

Severity

Informational

Description

`1714514461` from the `Location of Affected Code` section does not clearly state that it represents a date, associated with a check for the taxes.

Location of Affected Code

File: contracts/Portal/Portal.sol#L179

```
require(block.timestamp > 1714514461, "PORTAL:not yet.");
```

Recommendation

Consider defining the `timestamp` as a constant and give it a descriptive name, as that would improve code readability.

Team Response

Good call, will be fixed.

[I-09] Change Function Visibility from `public` to `external`

Severity

Informational

Description

It is best practice to mark functions that are not called internally as `external` instead, as this saves gas (especially in the case where the function takes arguments, as `external` functions can read arguments directly from `calldata` instead of having to allocate `memory`).

Location of Affected Code

Most smart contracts.

Recommendation

Consider changing the visibility of functions that are not used with the contract from `public` to `external`.

Team Response

Acknowledged, will be considered.

[I-10] Use `1e18` Instead of `10**18`

Severity

Informational

Description

It is recommended to use scientific notation (`1e18`) instead of exponential (`10**18`).

Location of Affected Code

File: contracts/Portal/modules/LiquidityModule/libs/LiquidityModuleLib.sol#L610-L612

```
if (supply > 0) {
  return (d * 10 ** 18) / supply;
}
```

Recommendation

It is recommended to change the code when initializing the variable and export that value as a constant.

Team Response

Good call, will be fixed.

[I-11] Use `require` Instead of `assert`

Severity

Informational

Description

The usage of the `assert` statement should be limited to testing internal errors and verifying invariants. Well-functioning code should never trigger a panic, even when encountering invalid external input. If such a situation occurs, it indicates a bug in your contract that requires fixing. Language analysis tools can assist in evaluating your contract to identify the specific conditions and function calls that may lead to panic.

Location of Affected Code

File: contracts/Portal/modules/StakeModule/libs/DepositContractLib.sol#L44

```
assert(_b.length >= 32 && _b.length <= 64);
```

File: contracts/Portal/modules/StakeModule/libs/DepositContractLib.sol#L68

```
assert(0 == temp_value); // fully converted
```

Recommendation

Consider using `require` statement instead.

Team Response

Acknowledged. However, we use assert to underline the conditions that should not fail. Will not be mitigated.

[I-12] Use a More Recent Solidity Version

Severity

Informational

Description

Currently, version `=0.8.7` is used across the whole codebase. Use the latest stable Solidity version to get all compiler features, bug fixes, and optimizations. However, when upgrading to a new Solidity version, it's crucial to carefully review the release notes, consider any breaking changes, and thoroughly test your code to ensure compatibility and correctness. Additionally, be aware that some features or changes may not be backward compatible, requiring adjustments in your code.

Location of Affected Code

All of the smart contracts use a relatively old solidity version.

Recommendation

Consider, upgrading all smart contracts to Solidity version `0.8.19`.

Team Response

Acknowledged, will be considered.

[I-13] Function Ordering does not Follow the Solidity Style Guide

Severity

Informational

Description

One of the guidelines mentioned in the style guide is to order functions in a specific way to improve readability and maintainability. By following this order, you can achieve a consistent and logical structure in your contract code.

Location of Affected Code

Most smart contracts.

Recommendation

It is recommended to follow the recommended order of functions in Solidity, as outlined in the Solidity style guide.

Functions should be grouped according to their visibility and ordered:

1. constructor
2. receive function (if exists)
3. fallback function (if exists)
4. external
5. public
6. internal
7. private

Team Response

Acknowledged. However, we do implement our own styling guidelines that closely resemble the official guidelines. We also accept that we need to improve the consistency of our guidelines in the future.

[I-14] Update External Dependency to the Latest Version

Severity

Informational

Description

Update the versions `@openzeppelin/contracts` and `@openzeppelin/contracts-upgradeable` to be the latest in package.json.

Location of Affected Code

According to package.json, `@openzeppelin/contracts` and `@openzeppelin/contracts-upgradeable` is currently set to 4.8.0.

Recommendation

I also recommend double-checking the versions of other dependencies as a precaution, as they may include important bug fixes.

Team Response

Acknowledged, will be considered.

[I-15] Missing/Incomplete NatSpec Comments

Severity

Informational

Description

(`@notice`, `@dev`, `@param` and `@return`) are missing in some functions. Given that NatSpec is an important part of code documentation, this affects code comprehension, audibility, and usability.

This might lead to confusion for other auditors/developers that are interacting with the code.

Location of Affected Code

In some contacts

Recommendation

Consider adding in full NatSpec comments for all functions where missing to have complete code documentation for future use.

Team Response

Acknowledged, will be considered.

[I-16] Open TODOs

Severity

Informational

Description

Open TO-DOs can point to architecture or programming issues that still need to be resolved. Often these kinds of comments indicate areas of complexity or confusion for developers. This provides value and insight to an attacker who aims to cause damage to the protocol.

Location of Affected Code

File: contracts/Portal/modules/GeodeModule/GeodeModule.sol#L169

```
// TODO: maybe seperate this? why not.
```

Recommendation

Consider resolving the TO-DOs before deploying code to a production context. Use an independent issue tracker or other project management software to track development tasks.

Team Response

Good call, will be fixed.

[I-17] Typos in Require Statement and Contract Comments

Severity

Informational

Description

In the following contract comments and require statements some typos were detected.

Location of Affected Code

spesific -> **specific**

Telescope is currently responsible from 4 tasks -> **Telescope is currently responsible for 4 tasks**

interpereted -> **interpreted**

while state is PROPOSED: validator proposed, it is passed, but haven't been created even tho it has been a MAX_BEACON_DELAY -> **while the state is PROPOSED: validator proposed, it is passed, but hasn not been created even though it has been a MAX_BEACON_DELAY**

it haven't been executed -> **it has not been executed**

operator have not used the withdrawal contract -> **operators have not used the withdrawal contract**

price kept same -> **the price is kept the same**

seperate -> **separate**

didnot -> **did not**

immidately -> **immediately**

Recommendation

It is recommended to correct the typos in contract comments and require statement messages.

Team Response

Thanks, we will fix those.

[G-01] Using `>>1` Instead of `/2` Can Save Gas

Severity

Gas Optimization

Description

A division by 2 can be calculated by shifting one to the right (>>1). While the DIV opcode uses 5 gas, the SHR opcode only uses 3 gas.

Location of Affected Code

File: contracts/Portal/modules/LiquidityModule/libs/LiquidityModuleLib.sol#L416

```
uint256 halfD = getD(xp, a) / 2;
```

File: contracts/Portal/modules/LiquidityModule/libs/LiquidityModuleLib.sol#L421

```
uint256 feeHalf = (dy * self.swapFee) / PERCENTAGE_DENOMINATOR / 2;
```

File: contracts/Portal/modules/LiquidityModule/libs/LiquidityModuleLib.sol#L581

```
v.feePerToken = self.swapFee / 2;
```

File: contracts/Portal/modules/LiquidityModule/libs/LiquidityModuleLib.sol#L850

```
uint256 feePerToken = self.swapFee / 2;
```

File: contracts/Portal/modules/LiquidityModule/libs/LiquidityModuleLib.sol#L1008

```
uint256 feePerToken = self.swapFee / 2;
```

Recommendation

Consider using shift right for tiny gas optimization.

/2 -> » 1

Team Response

Acknowledged, will be implemented.

[G-02] Using Booleans for Storage Incurs Overhead

Severity

Gas Optimization

Description

Booleans are more expensive than uint256 or any type that takes up a full word, because each write operation, emits an extra SLOAD to first read the slot's contents, replace the bits taken up by the boolean, and then write back. This is the compiler's defense against contract upgrades and pointer aliasing, and it cannot be disabled.

Location of Affected Code

Most smart contracts.

Recommendation

Use `uint256(1)` and `uint256(2)` for true/false instead.

Team Response

Acknowledged, will be reconsidered.

[G-03] Use Assembly to Check for `address(0)`

Severity

Gas Optimization

Description

Use assembly to check for `address(0)` to make the gas fees lower.

Location of Affected Code

Most smart contracts.

Recommendation

It is recommended to create a helper function that checks if the address is `address(0)` and use it in all the functions that are doing the `if(address(0))` check to reduce gas costs.

```
function assemblyOwnerNotZero(address _addr) public pure {
  assembly {
    if iszero(_addr) {
      mstore(0 x00 , "Zero address")
      revert(0 x00 , 0 x20 )
    }
  }
}
```

Team Response

Very good improvement, will be implemented.

[G-04] Splitting `require()` Statements that Use `&&` Saves Gas

Severity

Gas Optimization

Description

Instead of using the `&&` operator in a single `require` statement to check multiple conditions, using multiple require statements with 1 condition per require statement will save 8 GAS per `&&`. The gas difference would only be realized if the `revert` condition is met.

Location of Affected Code

Files:

- contracts/Portal/gETH.sol#L378-L383
- contracts/Portal/gETH.sol#L394-L399

Files:

- contracts/Portal/Portal.sol#L206
- contracts/Portal/Portal.sol#L208
- contracts/Portal/Portal.solL217

Files:

- contracts/Portal/modules/GeodeModule/libs/GeodeModuleLib.sol#L139-L142
- contracts/Portal/modules/GeodeModule/libs/GeodeModuleLib.sol#L198-L202
- contracts/Portal/modules/GeodeModule/libs/GeodeModuleLib.sol#L306

Files:

- contracts/Portal/modules/LiquidityModule/libs/AmplificationLib.sol#L102

Files:

- contracts/Portal/modules/LiquidityModule/libs/LiquidityModuleLib.sol#L296
- contracts/Portal/modules/LiquidityModule/libs/LiquidityModuleLib.sol#L508
- contracts/Portal/modules/LiquidityModule/libs/LiquidityModuleLib.sol#L1004

Files:

- contracts/Portal/modules/StakeModule/libs/DepositContractLib.sol#L44

Files:

- contracts/Portal/modules/StakeModule/libs/OracleExtensionLib.sol#L334

Files:

- contracts/Portal/modules/StakeModule/libs/StakeModuleLib.sol#L1379
- contracts/Portal/modules/StakeModule/libs/StakeModuleLib.sol#L1382
- contracts/Portal/modules/StakeModule/libs/StakeModuleLib.sol#L1484

Recommendation

Instead of using the && operator in a single require statement to check multiple conditions, use multiple `require` statements with 1 condition per `require` statement.

Team Response

Acknowledged, will be implemented.

[G-05] No Need to Initialize Variables with Default Values

Severity

Gas Optimization

Description

If a variable is not set/initialized, the default value is assumed (0, false, 0x0 ... depending on the data type). Saves 8 gas per instance.

Location of Affected Code

In all contracts where there is a for loop like this:

```
-- for (uint256 i = 0; ....
++ for (uint256 i; ....
```

Recommendation

Do not initialize variables with their default values.

Team Response

Acknowledged, will be implemented.

[G-06] Use `custom errors` Instead of `require` with Revert Strings

Severity

Gas Optimization

Description

Custom errors are available from Solidity version `0.8.4`. Custom errors save ~50 gas each time they are hit by avoiding having to allocate and store the revert string. Not defining strings also saves deployment gas.

Location of Affected Code

Most smart contracts.

Recommendation

Replace all require statements with Solidity `custom errors` for better UX and gas savings.

Team Response

Acknowledged, will be reconsidered.

[G-07] Expressions for `constant` Values Such as a Call to `keccak256()`, Should Use `immutable` Rather than `constant`

Severity

Gas Optimization

Description

Expressions that define a `constant` and involve calling a function are re-calculated each time the `constant` is referenced.

Location of Affected Code

File: contracts/Portal/gETH.sol#L49-L50

```
bytes32 public constant MIDDLEWARE_MANAGER_ROLE = keccak256("
    MIDDLEWARE_MANAGER_ROLE");
bytes32 public constant ORACLE_ROLE = keccak256("ORACLE_ROLE");
```

File: contracts/Portal/middlewares/ERC20PermitMiddleware.sol#L44-L45

```
bytes32 private constant _PERMIT_TYPEHASH =
  keccak256("Permit(address owner,address spender,uint256 value,uint256
      nonce,uint256 deadline)");
```

Recommendation

You could use `immutable` until the referenced issues are implemented, then you only pay the gas costs for the computation at deploy time.

Team Response

Very good improvement, will be implemented.

[G-08] Array Length Read in Each Iteration of the Loop Wastes Gas

Severity

Gas Optimization

Description

Reading array length at each iteration of the loop takes 6 `gas` (3 for mload and 3 to place memory_offset) in the stack. Caching the array length in the stack saves around 3 `gas` per iteration.

Location of Affected Code

Most smart contracts.

Recommendation

Cache the array length outside of the loop and use that variable in the loop.

Team Response

Very good improvement, will be implemented.

[G-09] Replace Constant Variables `public` Visibility with `private`/`internal`

Severity

Gas Optimization

Description

When constants are marked `public`, extra getter functions are created, increasing the deployment cost.

Location of Affected Code

File: contracts/Portal/gETH.sol#L49-L50

`public` -> `private`

```
bytes32 public constant MIDDLEWARE_MANAGER_ROLE = keccak256("
    MIDDLEWARE_MANAGER_ROLE");
bytes32 public constant ORACLE_ROLE = keccak256("ORACLE_ROLE");
```

File: contracts/Portal/modules/GeodeModule/libs/GeodeModuleLib.sol#L88-L93

`public` -> `private`

```
uint32 public constant MIN_PROPOSAL_DURATION = 1 days;
```

`public` -> `internal`

```
uint32 public constant MAX_PROPOSAL_DURATION = 4 weeks;
uint32 public constant MAX_SENATE_PERIOD = 365 days;
```

File: contracts/Portal/modules/LiquidityModule/libs/LiquidityModuleLib.sol#L97-L103

`public` -> `internal`

```
uint256 public constant MAX_SWAP_FEE = PERCENTAGE_DENOMINATOR / 100;
```

`public` -> `private`

```
uint256 public constant MAX_ADMIN_FEE = (50 * PERCENTAGE_DENOMINATOR) /
    100;
```

File: contracts/Portal/modules/StakeModule/libs/OracleExtensionLib.sol#L72-L78

`public` -> `private`

```
uint256 public constant MONOPOLY_RATIO = PERCENTAGE_DENOMINATOR / 100;
uint256 public constant MIN_VALIDATOR_COUNT = 50000;
uint256 public constant PRISON_SENTENCE = 14 days;
```

File: contracts/Portal/modules/StakeModule/libs/StakeModuleLib.sol

`public` -> `internal`

```
uint256 public constant MAX_GOVERNANCE_FEE = (PERCENTAGE_DENOMINATOR * 5)
    / 100;
```

`public` -> `private`

```
uint256 public constant MAX_MAINTENANCE_FEE = (PERCENTAGE_DENOMINATOR *
   10) / 100;
uint256 public constant MAX_ALLOWANCE = 10 ** 6 + 1;
uint256 public constant PRICE_EXPIRY = 24 hours;
uint256 public constant IGNORABLE_DEBT = 1 ether;
uint256 public constant MIN_VALIDATOR_PERIOD = 3 * 30 days;
uint256 public constant MAX_VALIDATOR_PERIOD = 2 * 365 days;
uint256 public constant SWITCH_LATENCY = 3 days;
```

File: contracts/Portal/modules/LiquidityModule/libs/AmplificationLib.sol#L23-L26

`public` -> `internal`

```
uint256 public constant A_PRECISION = 100;
uint256 public constant MAX_A = 10 ** 6;
uint256 public constant MAX_A_CHANGE = 2;
uint256 public constant MIN_RAMP_TIME = 14 days;
```

Recommendation

Marking these functions `private`/`internal` will decrease gas costs. One can still read these variables through the source code. If they need to be accessed by an `external` contract, a separate single-getter function can be used to return all constants as a tuple.

Team Response

Very good suggestion, will be reconsidered.

shieldify

Thank you!