GeodeFinance

Audit Response Report

March 2023

Prepared by 0xIcebear

Geodefi Ltd.

Contents

Introduction

This Report

This report is prepared to provide insight into the audit process and audit findings, implemented mitigations and further improvements; the effect of the review process on product development, and the post-audit period.

The main goal of this report is to provide a pleasant experience for future auditors, and document all the changes in the Protocol for reference.

This document consists of a short introduction, a summary of audit findings and mitigations, detailed explanations of the post-audit improvements, all contract changes with respective reasonings, and a short conclusion.

This document should take less than 2 hours to review.

This version of the report is finalized on March 22, 2023.

The Protocol

Geode Finance (The Protocol) is a Staking Protocol that allows anyone to create a Trustless Staking Pool (Pool). The principal objective of The Protocol is to remove the intermediaries within the staking environment by creating a **Global Standard** which provides security, accessibility, profitability, and improved user experience for the stakers.

The Protocol was founded as a public good in 2021 to respond to the monopolization of Liquid Staking Derivatives (LSDs). Extensive research was conducted to develop a decentralized infrastructure that directly connects the stakers with Industrial Node Operators without issuing trust among any of the parties. As a result, a secure approach for the pooling operations, validator creation process, withdrawal process, oracle operations, and smart contract upgradability patterns were developed.
Beta was launched on Avalanche Blockchain in June 2022.

The core of the protocol consists of a unique ERC1155 token named gETH, and a smart contract called Portal.
gETH acts as the central database of user balances and derivative prices. gETH also utilizes gETHInterfaces, permissioned smart contracts that provide functionality on top of the present data. gETH contract is **immutable**.
Portal handles pool creation and pool management, related staking operations such as user deposits and validator creations. Portal is **upgradeable** and utilizes the UUPS pattern.

Audit Process

The development process to bring The Procol to Ethereum started in February 2022. The Team created a very complex product for Ethereum Staking and finalized the codebase in October 2022.

To make sure that The Protocol will work flawlessly, and will not cause any problems for the users and user funds, the Team decided to work with Consensys Diligence on a comprehensive audit that will cover all the smart contracts within The Protocol.

The review was conducted over six weeks, from November 1st, 2022 to December 9th, 2022, by Sergii Kravchenko, Christian Goll, and Chingiz Mardanov (Auditors). A total of 60 person-days were spent.

The review was conducted on the commit hash:
8b07c0723d2a655a20d26620d4c3962cb9de4b00 .

During this period, the Team and the Auditors had regular meetings and asynchronous discussions where everything from simple variable naming to complex system designs were reviewed. Notably, the Auditors were very communicative, proficient in smart contract development, and unbothered by the complexity of the codebase.

This report was delivered on December 9th, 2022.

Post-Audit Period

While the audit was being conducted, the Team was already working on small fixes and implementing some improvements as a response to the feedback from the Auditors.

The Audit report underlined 21 issues, varying from Critical to Minor. Furthermore, the Auditors expressed numerous concerns during the review process, stating that the Protocol was not ready for deployment and contains many structural issues and uncertainties.

When the Audit report was received, the Development Team started discussing the design fallacies and shortcomings of The Protocol. Within this period, the Team had over 40 meetings with investors, potential partners, outside contributors, and community members; gathering feedback from different user groups. Moreover, 2 separate internal audits were conducted by different developers to identify the underlying issues within the codebase.

Following the audit report, meetings, and internal reviews, cumulatively more than 100 points were documented. These varied from simple suggestions to major bug fixes and breaking design proposals. Most consequential issues can be grouped, in order of importance:

1. Major Bugs
2. Unnecessary code complexity
3. Experimental implementations
4. Poor code readability
5. Complex terminology
6. Ambiguous trust assumptions
7. Unnecessary Oracle operations
8. Misplaced priorities
9. Unused code blocks
10. Lack of documentation
11. Vague product market fit

As a response, the development team was gathered in mid-January and declared an internal emergency period for the following 3 months, requesting all development resources to focus on protocol and smart contract development.

During this period, the vision behind The Protocol was heavily questioned. Previously, the Team was focused on creating *the best staking product* that will allow other smart contracts to utilize **auto-staking**, also known as superfluid staking. This was made possible by Dynamic Withdrawals supported by Liquidity Pools called *Dynamic Withdrawal Pools*. However, it was decided that **Dynamic Withdrawals was not a feature that can be simply implemented on Ethereum**. There is a consensus that this is the very idea that caused most of the issues within The Protocol.

The Staking Library

New approach is named **The Staking Library**, and it is exactly how it sounds like:

- The new architecture is **Modular**, allowing any functionality to be implemented within the Protocol without logical changes on Portal.
- The Protocol is still relatively complex, but the code is very simple and much smaller.
- Upgradability patterns are much stronger and more secure.
- Portal is less Governance dependent.
- Staking pools are now **configurable** and customizable.
- Staking pools are now **Permissionless!**
- Much better user experience for the staking pool owners.
- Near 100% test coverage.

In the most simple terms, The Protocol will provide the user experience the Beacon Deposit Contract can't. Geode will be the layer between Staking Pools and Node Operators.
Any arbitrary functionality that is possible for Staking, will be available with Geode. And users will choose how they want to combine and utilize them for their staking needs.

Implementation was finalized in early March. A version of The Protocol was deployed to Goerli and multiple node operators were onboarded.
Initial testing with the Node Operators is already done.
An immediate but not very comprehensive internal audit of the new approach was finished on March 16, 2023. The suggested mitigations are implemented.
A comprehensive internal audit will be finalized on April 8, before external audits.

In conclusion, the review process conducted by the Consensys Diligence Team effectively revolutionized Geode Finance.

Summary

Summary

Contract Name	Type	Status	Old Version	New Version
globals	Internal Library	Added	-	Here!
gETH	Contract	Small Improvement	Here!	Here!
ERC20InterfaceUpgradable	gETHInterface	Small Improvement	Here!	Here!
ERC20InterfacePermitUpgradable	gETHInterface	Small Improvement	Here!	Here!
MathUtils	Internal Library	No changes	Here!	Here!
AmplificationUtils	Internal Library	No changes	Here!	Here!
SwapUtils	External Library	Clean Up	Here!	Here!
LPToken	Contract	No changes	Here!	Here!
Swap	Contract	Clean Up	Here!	Here!
MiniGovernance	Contract	Removed	Here!	-
WithdrawalContract	Contract	Added	-	Here!
DepositContractUtilsLib	Internal Library	Refactor	Here!	Here!
DataStoreUtilsLib	Internal Library	Improvement	Here!	Here!
GeodeUtilsLib	External Library	Clean Up	Here!	Here!
MaintainerUtilsLib	Internal Library	Removed	Here!	-
OracleUtilsLib	External Library	Breaking	Here!	Here!
StakeUtilsLib	External Library	Breaking	Here!	Here!
Portal	Contract	Clean Up	Here!	Here!
IDepositContract	Interface	Refactor	Here!	Here!
IERC20InterfacePermitUpgradable	Interface	Removed	Here!	-
IgETHInterface	Interface	Added	-	Here!
IGeodeModule	Interface	Added	-	Here!
IWhiteList	Interface	Added	-	Here!

IMiniGovernance	Interface	Removed	Here!	-
IWithdrawalContract	Interface	Added	-	Here!
IPortal	Interface	Refactor	Here!	Here!
ILPToken.sol	Interface	Refactor	Here!	Here!
ISwap	Interface	Clean Up	Here!	Here!
IgETH	Interface	Refactor	Here!	Here!
OwnerPausableUpgradeable	Helper	No changes	Here!	Here!
TestMathUtils	Helper	No changes	Here!	Here!
TestSwapReturnValues	Helper	No changes	Here!	Here!
TestDepositContractUtils	Helper	Refactor	Here!	Here!
TestDataStoreUtils	Helper	Improvement	Here!	Here!
TestGlobals	Helper	Added	-	Here!
TestGeodeUtils	Helper	Removed	Here!	-
TestStakeUtils	Helper	Removed	Here!	-
WhiteList	Helper	Added	-	Here!
ERC1155ReceiverMock	Helper	Refactor	Here!	Here!
nonERC1155Receiver	Helper	Refactor	Here!	Here!
DepositContract	Helper	Removed	Here!	-

1. General

a. Even though this version is far more functional, most of the changes are just removing big code chunks or renaming concepts/variables/files.

b. globals.sol is added to improve code readability. It includes global parameters or enums such as PERCENTAGE_DENOMINATOR, ID_TYPE, VALIDATOR_STATE.

c. gETH avoiders logic is now ID specific, as advised.

d. A solidity interface named IgETHInterface was introduced, providing a standard for all gETHInterface contract initializers. Allowing the Team to generalize the approach on initiatePool.

e. IGeodeModule added, providing a standard upgradability logic for any Geode Module. Only utilized by Portal and WithdrawalContracts for now. This contributes to modular architecture.

f. **Dynamic Withdrawal Pool is renamed to Liquidity Pool.**
g. SwapUtils and Swap.sol now enforces 2 tokens for gas improvement.
h. Tests for the external libraries - GeodeUtils, StakeUtils, OracleUtils - are conducted through Portal instead of test contracts as a response to an issue.
i. maintainerUtils library is removed. Some functionality moved into StakeUtils.
j. Near 100% code coverage is achieved.

2. Codebase and Modular Architecture

a. DataStore struct is renamed as **IsolatedStorage**.
b. Array logic with getters, appenders, and batch appenders are introduced to DataStoreUtils:
    i. Every array has a key (arrayKey), that is generated by getKey(id, “name”).
    ii. Data pointer for an element is getKey(index, arrayKey)
    iii. The length of an array is simply self.uintData[arrayKey]
    iv. Batch array appenders should be used to save gas cost
c. Added all readDataStore (view) functions to Portal and removed batch getters like getPlanet.
d. IGeodeModule has a circuit breaker, named **recoveryMode**. Which signals other contracts to stop relying on them:
    i. WithdrawalContract checks Portal before fetching an upgrade.
    ii. Portal checks withdrawalContract before proceeding in deposit or proposeStake functions.
    iii. Current recoveryMode conditions for Portal:
        1. Contract is paused
        2. Contract needs to be upgraded or initialized
        3. Senate expired, which is possible.
    iv. Current recoveryMode conditions for Withdrawal Contract:
        1. Contract is paused
        2. An upgrade needs to be fetched from Portal
        3. Contract needs to be upgraded or initialized
        4. Pool owner and Contract owner are different
        5. Senate expired, which is not possible.
e. **Default module** and **Allowed module** logics are added, as advised.

f. newProposal checks for LIMIT parameters of globals.sol to decide if a module is “default” or “allowed”:
    i. A Module is distinguished by its TYPE. Within ranges determined in globals.sol, it is either a defaultModule or an allowedModule.
    ii. There is *only 1 instance* of a defaultModule:
        1. Withdrawal Contract
        2. Liquidity Pool
        3. Liquidity Pool Token
    iii. There can be *many instances* of an allowed module:
        1. gETHInterfaces

g. **fetchUpgradeProposal** logic is improved.
    i. Previously:
        1. Anyone can fetch a proposal but can not approve. Very **Faulty,** susceptible to griefing attacks if timed well.
        2. It reads the default withdrawal contract version via Portal.
        3. The Withdrawal Contract reads a proposal from Portal. **Faulty**; proposals are not in Portal to “*be read”*.
        4. Constructs a new proposal with checks and stuff. **Faulty**, susceptible to griefing attacks if a wrong contract is provided to begin with.
        5. Forgets calling upgradeTo and doesn’t even go under recoveryMode. Well, **faulty**.
    ii. Now:
        1. Withdrawal Contract calls Portal.fetchModuleUpgradeProposal function giving its TYPE.
        2. Withdrawal Contract checks if Portal is in recoveryMode before making a call.
        3. Portal.fetchModuleUpgradeProposal creates a proposal in the withdrawal contract, msg.sender, since newProposal is a part of IGeodeModule logic.
        4. Just approves and calls upgradeTo immediately.

3. Governance

a. Universe struct is renamed as **DualGovernance**.

b. Senate Election logic, related parameters, functions, and events are removed.

   i. Instead, Senate will be an external contract that handles how proposal approvals work internally.

   ii. If Senate change is needed, it can be achieved with a proposal TYPE 1 or **globals.ID_TYPES.SENATE**

   iii. Governance can choose to keep the same Senate active: Proposal CONTROLLER would be the current Senate.

   iv. Governance can choose another contract as the Senate, which is effective if the current Senate approves.

   v. Governance should propose a new Senate before Senate Expiry. Otherwise, it is advised that users should stop using The Protocol.

   vi. If Senate ever expires, Portal goes under recoveryMode, which doesn't change any internal functionality.

   vii. **rescueSenate** function, a circuit breaker utilizing Senate Expiry is introduced.

c. FEE_COOLDOWN is introduced. Not really important.

d. A function like **updateStakingParams** is not needed anymore, because there are barely any parameters to update. Reducing the direct control and effect of the Governance.

4. Staking Pools

a. StakePool struct is renamed as **PooledStaking**. Effectively, its only remaining parameter from the old version is gETH.

b. Pools can not be Operators anymore.

c. Pools are **permissionless.** However, it requires at least 32 Ether, 1 validator worth of funds, to prevent sybil attacks.

d. Planet - Comet separation is merged under **Configurable Staking Pools.**

e. Pool Customizability and Modules:

   i. **gETHInterface**: Optional. Can choose any Interface that is allowed. **Can NOT be changed after initiation**. Utilized as an allowed module.

    ii. **Bound Liquidity Pool**: Optional. Can choose to deploy or not. Can be deployed later. Can be replaced with a new version later. Utilized as a default module.
    iii. **Withdrawal Contract**: Mandatory. Can choose to upgrade to a new contract later. Utilized as a default module.
    iv. **Visibility**: Can be private or public. Can use a whitelisting contract if private. Not a module.
    v. **Maintainers**: Optional. Can be used to automate daily operations. Currently, validator distribution via batchApproveOperators for pools and other maintenance. Can be used by Operators for proposeStake and beaconStake operations. Not a module.

f. Dynamic Withdrawal Logic is completely removed from the StakeUtils, including functions and parameters like withdrawalBoost, donateBalancedFees, withdrawPlanet, etc.
g. Issues around the implementation of **surplus burning** were identified and it was removed. Simply, it was problematic for the Validator creation process.
h. pauseStakingForPool and unpauseStakingForPool functions are removed, now staking can be stopped by pausing pool's withdrawal contract and pushing it into **recoveryMode**.
i. Better and **much clearer Initiators**.

5. Oracle

a. **Superior simplification**.
b. Oracle's struct is no longer a parameter in stakeUtils' struct. OracleUtils doesn't even have a struct anymore.
c. Prevent delegateCall hell. Instead, Oracle uses StakeUtils if needed. StakeUtils doesn't even know OracleUtils exists.
d. No more maintaining the **time constraints**. Previously worked on the first 30 min of a given day, where some functions such as beaconStake was deactivated.
e. Because of the above point, and other issues that identified with its methodology, **mint/burn buffers are removed**.
f. Oracle operations are **simplified**. No more finding the real price but confirming some other price by using some data from The Beacon Chain -while pretending to not trust it-, and utilizing some "buffers"

which doesn’t even work because you can never know who is burning tokens and when.

    i. Oracle works multiple times a day.
        1. Approving Validator Proposals every **x** hours, and every **y** proposals.
        2. Updating price merkle root every **z** hours, and if any derivative price changes more than **k**%.
        3. Operators are regulated if and when there is an issue.
    ii. Oracle Nodes calculate the prices of all derivatives and create a merkle root.
    iii. If consensus is achieved, merkle root and validator proposals are sent.
    iv. When a new price merkle root is set, all validator prices are invalidated and deposit/withdrawal operations are deactivated until the price is validated with proper merkle proofs.
    v. Someone who wants to mint new tokens needs to call priceSync.
    vi. *deposit* function takes merkle proofs and operates a priceSync, if the price is invalid.
    vii. If there are no Oracle updates within **H** hours, all prices are invalidated.

g. To make things easier for the stakers, isPriceValid checks are added when *price* is needed such as in the ***deposit*** **function which takes merkle proofs** and operates a priceSync if the price is invalid. This way, you can make sure that there is **no cheap minting**!
h. Clean up for unrelated functionality. Mostly moved into StakeUtils.

6. Withdrawal Credential Contract

a. Inherites IGeodeModule.
b. **Previously MiniGovernance, renamed to WithdrawalContract**.
c. IsolationMode is replaced and improved with recoveryMode from IGeodeModule.
d. Unnecessary passwordHash logic is removed, instead, recoveryMode is utilized to check if pool owner is not the contract owner.
e. Calling refreshSenate every now and then to signal activity is deprecated. Withdrawal Contract’s **Senate never expires**.
f. claimUnstake logic is removed. Funds never return to Portal once validators are created.
g. PoolGovernance struct is removed, no more needed.

Audit Mitigations

Audit Findings

Scope								
File Name	SHA-1 hash	Type	# Findings				Findings	**Status (pass)**
			🟥	🟧	🟨	⬛		
MiniGovernance.sol	3ef6d711e	Contract	0	1	3	0	4, 7, 9, 15	🟥
Portal.sol	3e2fddb0a	Contract	0	1	1	0	4, 8	🟥
gETH.sol	0c149c8ee	Contract	0	1	1	0	3, 6	🟨
ERC20InterfacePermitUpgradable.sol	1aa5cf595	gETHInterface	0	0	0	0	-	🟩
ERC20InterfaceUpgradable.sol	8bf2ca0ab	gETHInterface	0	0	0	0	-	🟩
ERC1155SupplyMinterPauser.sol	d228841d7	Helper	0	0	0	0	-	🟩
DataStoreUtilsLib.sol	680b86043	Internal Library	0	0	0	1	21	🟩
GeodeUtilsLib.sol	db36c1cd3	External Library	0	0	0	0	-	🟩
MaintainerUtilsLib.sol	f171c4ea1	Internal Library	0	0	1	1	14, 18	🟥
OracleUtilsLib.sol	7b606059b	External Library	2	0	3	0	1, 2, 5, 16, 17, 19	🟥
StakeUtilsLib.sol	dd33fa886	External Library	1	1	8	2	2, 3, 6, 7, 9, 10, 12, 13, 14, 16, 19, 20	🟥
LPToken.sol	66124ef6d	Contract	0	0	0	0	-	🟩
Swap.sol	ca1fad6d3	Contract	0	0	0	0	-	🟩
AmplificationUtils.sol	f5afa0fb6	Internal Library	0	0	0	0	-	🟩
MathUtils.sol	897743675	Internal Library	0	0	0	0	-	🟩
SwapUtils.sol	2736852fd	External Library	0	0	0	0	-	🟩

1. Oracle’s _sanityCheck for prices will not work with slashing

Severity: Critical.
Mitigated, Improved.

The new approach to the oracle price updates is slightly different. Previously, Oracle would conduct the sanity check and proceed to update the prices of gETH derivatives. Now, Oracle only sends a Merkle Root. When a new Root is sent, isPriceValid returns false for all pools, invalidating all previous gETH prices.
deposit Function in the StakeUtils checks for isMintingAllowed, which checks for price validity.
deposit function in Portal accepts MerkleProofs, updating the derivative price if needed.
If there are no Root updates for 24h, the price is invalidated automatically.
Furthermore, Oracle has a slight delay before publicizing the proofs, meaning no one can see them until the Root is finalized.
Currently, there are no ways to mint cheap tokens by frontrunning the Oracle.

Secondly, if the price falls below a certain threshold, currently 7% daily, the price can not be updated. Meaning if a validator is slashed, price can not be updated for the next 7 days and no more deposits are allowed for this pool within this period.
Same goes for price increases but it is expected that threshold will be lower than 7%.
This is not a bug, this is a feature, a circuit breaker.

Finally, _sanityCheck function had an underflow issue detected, which is also mitigated.

2. Multiple calculation mistakes in the _findPricesClearBuffer function

Severity: Critical.
Deprecated.

_findPricesClearBuffer is deprecated. It is worth noting that the Team also identified a bug that invalidates the whole logic of buffers.

3. New interfaces can add malicious code without any delay or check

Severity: Major.
Mitigated, Improved.

Thanks to Modular Architecture, the Protocol utilize _allowedModules for gETHInterfaces now. This effectively means any interface is, for sure, proposed by the Governance and approved by the Senate. The Team agrees that user security is much more important than developer experience.

Additionally, gETH avoiders logic is changed to be ID based rather than users avoiding all interfaces.

4. MiniGovernance - fetchUpgradeProposal will always revert

Severity: Major.
Mitigated, Improved.

The hard-coded proposal duration is switched to 3 weeks.
Notably, the fetch-upgrade pattern is also improved with the new function that is used in Portal: fetchModuleUpgradeProposal. This function operates for any modules that utilize IGeodeModule.

5. reportOracle can be sandwiched for profit.

Severity: Medium.
Mitigated, Improved.

The explanation for issue 1 applies here as well. No one can mint cheap tokens. However, yes, they can transfer it or trade it, which is out of scope for us.

6. Updating interfaces of derivatives is done in a dangerous and unpredictable manner.

Severity: Medium.
Mitigated.

The explanation for issue 3 applies here as well.
Additionally, a pool can be configured with any interface that is allowed, but Portal doesn't allow changing the interface address or using multiple interfaces.

7. A sandwich attack on fetchUnstake

Severity: Medium.
Deprecated.

fetchUnstake was a part of the Dynamic Withdrawal Pools functionality and DWP is deprecated.

It is worth noting that, now, Portal will never touch the validator funds after they are sent to the beacon chain. All the exit operations will be handled by the withdrawal contract.

8. Only the GOVERNANCE can initialize Portal

Severity: Medium.
Deprecated.

There is no longer a function called updateStakingParams because there are no parameters for Governance to update. This is a great improvement, minimalizing the Governance dependency, and making Portal much stronger in terms of trustlessness.

9. The maintainer of the MiniGovernance can block the changeMaintainer function.

Severity: Medium.
Deprecated, Improved.

Maintainer - Controller separation is now very well defined in StakeUtils:

- CONTROLLER: owns and controls.
- maintainer: automates, optimizes, and serves.

WithdrawalContract's owner is the pool's Controller. There is a possibility that the two can be different if not changed simultaneously. In that case, the withdrawalContract will go under recoveryMode, signaling Portal to seize minting and staking operations.

It is worth noting that even if a maintainer goes rogue and effectively imprisons its operator, it can be changed by the CONTROLLER. Because changeMaintainer is not secured with the authenticate function, but has local checks.

10. Entities are not required to be initiated.

Severity: Medium.
Mitigated.

authenticate function checks for *initiated* parameter for both operator and pool TYPEs.

11. Node operators are not risking anything when abandoning their activity or performing malicious actions

Severity: Medium.
Acknowledged.

This is a design choice that is well expressed.

Notably, Geode is much better than its competitors. While some staking pools distribute fees every day and effectively risk paying a fee for no real returns, Geode distributes Operator and Pool owner fees if and when the funds are back in the Execution Layer - withdrawalContract of the pool -.

12. Planets should not act as operators

Severity: Medium.
Mitigated.

Planets, renamed as *Pools*, do not act as operators anymore.

13. The blameOperator can be called for an alienated validator

Severity: Medium.
Mitigated.

blameOperator now checks if the validator is active instead of checking if the validator is not exited.

14. Latency timelocks on certain functions can be bypassed.

Severity: Medium.
Mitigated.

Latency timelocks on both *fee* and *validatorPeriod* are improved to prevent any function calls when it's locked. Once the parameter starts switching, it can not be changed again.

15. MiniGovernance’s senate has almost unlimited validity

Severity: Medium.
Mitigated.

WithdrawalContracts, previously miniGovernances, do not utilize senate expiry anymore. The logic of refreshing the senate expiry was introduced with the concerns of a pool owner leaving their pool. With the current set up this is not a problem and withdrawal contracts do not require such action. Funny enough, withdrawalContracts certainly have unlimited validity now.

16. Proposed validators are not accounted for in the monopoly check.

Severity: Medium.
Mitigated.

totalProposedValidators and *proposedValidators* parameter for operator TYPEs is introduced.
It is increased when a validator is proposed.
It is decreased when a validator is either activated or alienated.
It is checked before proposing new validators via proposeStake.

17. Comparison operator used instead of assignment operator

Severity: Medium.
Mitigated.

18. initiator modifier will not work in the context of one transaction.

Severity: Medium.
Deprecated, Improved.

MaintainerUtils is deprecated. Likewise, the modifier initiator is removed. However, StakeUtils still has initiators, but these initiators are easy to understand, designed with *customizability*.

It is now very easy to understand what initiatePool and initiateOperator functions are doing.

19. Incorrect accounting for the burned gEth.

Severity: Medium.
Deprecated.

Buffers are removed.

20. Boost calculation on fetchUnstake should not be using the cumBalance when it is larger than debt.

Severity: Medium.
Deprecated.

fetchUnstake is removed.

21. DataStore struct not having the _gap for upgrades.

Severity: Medium.
Mitigated.

_gap[12] is added to the DataStore struct, renamed as IsolatedStorage.

Audit Notes

As a response to the Code Quality Recommendations:

- Introduced the globals.sol file which is used to define the arbitrary parameters within The Protocol. Using UINT parameters with well-defined values was preferred over using Enums.

- MaintainerUtils is removed. *maintainer* logic was revised and moved into StakeUtils with a relatively small functionality.

- WithdrawalContract, previously miniGovernance, using a proposal to fetch an upgrade proposal is fixed. IsolatedStorage is a holy place for us and the Proposals should not be used in any other way than read functionalities.

- Authentication issue was resolved. Every contract has well-placed access checks:
    - geodeUtils has OnlyGovernance, OnlySenate and OnlyController checks with modifiers.
    - stakeUtils has the "authenticate()" function which checks for Maintainers, Controllers, and TYPE. Except changeMaintainer, it has its own local checks.
    - oracleutils has OnlyOracle checks with a modifier.
    - Portal has an OnlyGovernance check on: pause, unpause, pausegETH, unpausegETH, setEarlyExitFee, releasePrisoned.

- As you can see, the naming convention is heavily revised and will be continued.

Post Audit Findings

1. Faulty Validator Creation

Severity: Critical.

Signature is one of the parameters while calling the Beacon deposit contract to initiate a validator. Normally, this signature is created by a CLI provided by Ethereum.
proposeStake function takes a signature parameter from the operator and uses it to send 1 Eth to initiate the validator. Then, it is stored in the validator struct. When the beaconStake function is called by the operator; the very *same signature* is used again, sending 31 ETH to the previously approved validators.

However, signature creation also takes the amount of Ether being sent into consideration. This means using the same signature will fail either way.

Mitigation:
proposeStake takes 2 signatures:

- Sig1: used on proposeStake while sending the Ether to the deposit contract.
- Sig31: stored in the validator struct to be used in the beaconStake calls.

Both parameters are validated between proposeStake and beaconStake operations by Oracle while updating the Verification Index. But just the Sig31 is stored in the validator struct.

2. Dynamic Withdrawals do not work.

Severity: Critical.

Dynamic withdrawals allow node operators to profit from the arbitrage between the principal and its price. Thus, creating very healthy derivatives that are sustained by balancing the supply and demand between the consensus layer and the execution layer.

Simply put, Ethereum staking doesn't have instant validator exits, making it impossible to capture the arbitrage for the operator. Because there is a withdrawal queue, according to the EIP-4895, and it can take days for exited funds to be withdrawn.

Since this arbitrage is the core of this mechanism, it is not possible to implement Dynamic Withdrawals for Ethereum staking.

Mitigation:
Dynamic Withdrawal functionality is removed. Related functions and parameters such as fetchUnstake, signalUnstake, switchWithdrawalBoost, withdrawalBoost, BOOST_SWITCH_LATENCY, and related events are removed.
Instead, all Pools will have withdrawal queues that are handled in the withdrawalContract. Furthermore, with this fix Portal does not have special permissions on withdrawalContract. Previously, it could drain any amount of Ether without any checks...

3. Inaccurate assumption on buffers used in oracle price calculation

Severity: Critical.

Telescope was designed to work once every night, but it can not provide data while still collecting it. So, there is a delay between calculation and registration. Because of this, Portal utilizes mint and burn buffers. When the oracle is working for the first 30 minutes of the day, buffers track the supply change.

There is a false assumption that the burn buffer is tracking the balance decreases correctly while burning tokens from the surplus. However, anyone can burn their own tokens without bothering Portal.

Because of this, Portal will not find the correct oracle price and will not be able to prove it from the Merkle Root. Thus, simple 1 gwei burning at the correct time can prevent updating the derivative prices and Merkle Root.

Mitigation:
First of all, Oracle works all the time, it doesn't have these limitations anymore.

Secondly, Oracle only updates the Merkle Root and does not issue any calculations or checks.
Finally, the derivative price will be updated by the user when it is going to be used. If it is not updated, minting operations are not allowed.

In conclusion, buffers are removed as well as parameters and functions such as ORACLE_PERIOD, ORACLE_ACTIVE_PERIOD, _isOracleActive checks, etc.

4. Missing fallback functions

Severity: Major.

Simply, Portal and Withdrawal Credential Contracts don't have fallback functions.

Mitigation:
Internally, the reasoning behind this issue is detected as "test contracts" of external libraries. Portal has a lot of tests but mostly all of them utilize these test contracts. So, even though coverage seemed good, we were testing with contracts that contain fallback functions.
Now, all external libraries of Portal and WC are tested utilizing the real contracts. Related test contracts are removed.

5. Maintainer can attack the Node Operator

Severity: Major.

Differently from the 9th issue of the Diligence report, a maintainer can uniquely harm its owner.
When a faulty validator is proposed, the operator ID is imprisoned meaning the owner's access to the Portal is very limited. Thus, the Operator's CONTROLLER can not change the maintainer and the maintainer can continue attacking effectively making the ID useless forever.

Mitigation:
changeMaintainer function does not use *authenticate* function to prevent the effects of any additional logic. Instead, it checks if the caller is CONTROLLER.

6. Fee donation may cause some serious problems

Severity: Major.

As a security measure, being able to manipulate the Liquidity Pool parameters should be avoided. Furthermore, the fee donation process does not take the market price but the oracle price into account, which makes things very difficult to confirm.
Finally, this function is not only used by Portal while burning surplus, it is open to access for everyone. There is a potential risk that price will be somehow manipulated. This risk should not be taken by The Procol.

Mitigation:
Burning surplus is not allowed anymore.

7. Burning surplus causes disruption

Severity: Major.

Portal allows stakers to redeem collateral directly from not-yet-staked funds. This is a good user experience. However, there are no queues or timelocks, making things very hard for Operators.
An Operator can call proposeStake with 50 validators worth of data. At the end of the day, it takes *burning one wei worth of surplus* for this transaction to be grieved.

Mitigation:
Burning surplus is removed.
donateBalancedFees function from SwapUtils is removed. _burnSurplus, _donateBalancedFees, and withdrawPlanet functions from StakeUtils are removed.
Additionally, this is very good because it introduced some other vulnerabilities by donating fees, allowing frontrunners, collateral loss to arbitrage, etc.

8. Sanity check potential underflow issue

Severity: Medium.

Sanity check allows up to x% daily decrease in the price taking the last update timestamp as a reference. Here is the calculation:

```
uint256 minPrice = curPrice -
          ((curPrice *  self.PERIOD_PRICE_DECREASE_LIMIT *
            _periodsSinceUpdate) / PERCENTAGE_DENOMINATOR);
```

It is obvious that, if _periodsSinceUpdate is very long this calculation will start underflowing, effectively making it impossible to update the prices again.
For example, if x is 5%, the permitted price decrease will be bigger than *curPrice* after 20 days:

x-((x*5*20)/100) = 0;

Taking the new(current) oracle logic into consideration, this calculation should be handled with care because there are no enforced oracle updates and a pool can choose to keep the old price until the end of the time.

Mitigation:

```
uint256 maxPriceIncrease = ((curPrice *
 STAKER.DAILY_PRICE_INCREASE_LIMIT *
 dayPercentSinceUpdate) / PERCENTAGE_DENOMINATOR) / PERCENTAGE_DENOMINATOR;

uint256 maxPriceDecrease = ((curPrice *
 STAKER.DAILY_PRICE_DECREASE_LIMIT *
 dayPercentSinceUpdate) / PERCENTAGE_DENOMINATOR) / PERCENTAGE_DENOMINATOR;

require(
 (_newPrice + maxPriceDecrease >= curPrice) && (_newPrice <= curPrice + maxPriceIncrease),
 "OU: price is insane"
);
```

9. Use _disableInitializers function

Severity: Medium.

Openzeppelin UUPS is utilized by Portal and WithdrawalContract. Furthermore, by using clones, TransparentUpgradeableProxy is utilized by gEthInterfaces: ERC20InterfaceUpgradable and ERC20InterfacePermitUpgradable.

Openzeppelin introduced **an issue** specifying that uninitialized implementation contracts can cause a lot of problems.

Mitigation:
_disableInitializers() function is added to the constructors of Portal, withdrawalContract, ERC20InterfaceUpgradable, and ERC20InterfacePermitUpgradable.

10. depositPlanet might fail because of swap(mingETH)

Severity: Medium.

Simply, when a user calls the depositPlanet function of StakeUtils, mingETH is provided as a parameter. mingETH represents a minimum on the returned gETH amount. This limit applies to the total of minted and market-bought tokens. However, the same parameter is passed to DWP if there is a debt. This call can fail on partial buyback, even though minimum requirements would be met at the end of the call.

Mitigation:
mingETH is used only at the end of the deposit operation, in a requirement state comparing it with the gETH amount user will receive.

11. Withdrawal contract upgrades can be manipulated

Severity: Medium.

The function to fetch the latest version of the Withdrawal Contract can be called by anyone. There are 2 most important issues this will cause:

1. Proposals expire, meaning if a pool controller wasn't aware of the fetch operation, it can never be upgraded to the latest version.
2. Fetching a proposal can be used to stop the pooling operations. Whenever an upgrade proposal is fetched, it should be immediately approved. And only the Senate, which is the pool owner, can approve a proposal.

Mitigation:
The pattern to upgrade the withdrawal contract is completely redesigned:

- IGeodeModule is introduced, which consists of newProposal, approveProposal, recoveryMode, and isUpgradeAllowed.
- Both Portal and Withdrawal Contract inherit IGeodeModule, allowing them to communicate and operate smoothly.
- When a fetchModuleUpgradeProposal function from the Portal is called by any module, Portal takes the current defaultModule for the given TYPE and calls newProposal on the caller.
- Then, the proposal can be accepted or rejected by the caller.
- In case of the Withdrawal Contract, fetchUpgradeProposal should be called by the Owner.
- Proposal is approved and the contract is upgraded immediately after the fetch operation.
- Notably, fetchUpgradeProposal checks for Portal's recoveryMode before getting in touch with the Portal, as it was intended.

With this setup, only the Owner can call the upgrade proposal and there is no time spent between fetch and upgrade.

12. Portal can not pause or unpause gETH

Severity: Medium.

gETH has Pausability: pause and unpause. However, Portal does not have respective functions for Governance to use them. These circuit breakers can be useful in case something went wrong.

Mitigation:
Mentioned functions are added to the Portal with onlyGovernance modifier protection.

13. Potential Governance Attacks

Severity: Medium.

While the project is designed and marketed with the core idea of Limited Upgradability and Dual Governance, which in effect means that Governance can not attack, harm, or grieve the smart contracts without the permission of a Senate, this is not completely accurate.
Portal has a updateStakingParams function which allows Governance to change some parameters. Note that these parameters are very important and **not** ignorable or well-protected.

1. _DEFAULT_gETH_INTERFACE can be changed to steal user funds.
2. _MAX_MAINTAINER_FEE can be decreased to 0% without any checks effectively removing the functionality.
3. _PERIOD_PRICE_INCREASE_LIMIT or _PERIOD_PRICE_DECREASE_LIMIT can be decreased to 0%, effectively preventing the price updates.

Mitigation:
All these parameters and many more are already removed from StakeUtils and OracleUtils libraries. There is no need to have an updateStakingParams function anymore, and it is removed.
Now, governance can not set any crucial parameters without upgrade proposals.

14. Faulty alienation

Severity: Medium.

When a validator is alienated, changes on the pool variables surplus and secured are reverted. However, some parameters like *proposedValidators* are not changed.

Similarly, proposeStake includes the validator in the pool's "*validators*" array. This would be ok if there was a way to remove the elements from arrays. Since there is no way to revert this change in case of a faulty validator proposal, it should be included in *beaconStake* function.

Mitigation:
_alienateValidator function is fixed and checks for respective parameters now.
Validator pubkey is added to the pool's array on beaconStake.

15. Utilize gETH avoiders in the contracts

Severity: Minor.

gETH avoiders logic is designed to give any address the freedom to protect itself from malicious interfaces. However, the Team is not using avoiders in their contracts and not following their own best practices.
Portal, Liquidity Pool, and Withdrawal Contracts can and must have some gETH at some point in their operations. And there is no reason for an interface to use them, as there is no ERC20 functionality.
Protect your user funds.

Mitigation:
Both Portal and Swap contracts have avoiders now.
Liquidity Pool avoids when the contract is initialized.
Portal avoids when the Pool is initialized.

16. Swap.sol should use fixed arrays to save gas

Severity: Minor.

Self-explanatory.

Mitigation:
Swap and SwapUtils now utilize static arrays with length 2.

17.Using a password and hash is not vulnerable, but inappropriate

Severity: Minor.

Self-explanatory.

Mitigation:
Maintainer - Controller separation is now very well defined in StakeUtils:

- CONTROLLER: owns and controls.
- maintainer: automates, optimizes, and serves.

WithdrawalContract's owner is the pool's Controller. There is a possibility that the two can be different if not changed simultaneously. In that case, the withdrawalContract will go under recoveryMode, signaling Portal to seize minting and staking operations.

18. DataStore is not accessible

Severity: Minor.

The idea behind the DataStore is to provide easy and manageable read-write operations. However, it seems like the team forgot this and implemented batch read functions such as getPlanet, getComet, getOperators, etc.
There are even single functions for parameters like fee, validatorPeriod, etc.

Mitigation:
Read functions are now exposed in the Portal.
Other related single and batch read functions are removed.

19. High MONOPOLY_RATIO

Severity: Minor.

Portal aims to prevent Monopolies within its operator marketplace, thus any operator with more validators than a certain threshold can not create additional validators.
However, this parameter is set to 5%. Which is not remotely even close to preventing monopolies.

Mitigation:
MONOPOLY_RATIO parameter is now 1%.

20. Flawed Upgradability Pattern

Severity: Info.

When Portal or WithdrawalContract is upgraded to a newer version, it is advised to set DualGovernance’s approvedUpgrade to address(0). Notably, implemented initialize functions contain this logic.
This would be a small issue, but some dire consequences surface if this advice is not followed.

Mitigation:
The method of _setContractVersion which is included by both contracts, is modified.
Additionally, *isUpgradeAllowed* function of geodeUtils now checks for proposed implementation and current implementation. Address of the current implementation can always be reached with UUPS._getImplementation().

21. Flawed Architecture: Nested Libraries

Severity: Info.

StakeUtils and OracleUtils:
Expected behavior: StakeUtils to contain the functionality around pooling and staking operations and OracleUtils to contain the functionality around the oracle operations.
However, OracleUtils is effectively nested within StakeUtils. StakeUtils struct has a parameter called Telescope, which is the struct of OracleUtils.
OracleUtils includes unrelated functions like _canStake, while Stakeutils has a function with the same name. These functions are doing different things...
The relation between these two libraries is obviously positioned wrong and causes some problems within the codebase.

Mitigation:

OracleUtils struct is removed. The Library is cleaned up of unnecessary complexity, and unrelated functionality and downgraded into 3 functions and its internal helpers.
OracleUtils is importing StakeUtils now. Effectively, StakeUtils is not even aware that OracleUtils exist.

22. Flawed Architecture: Arbitrary Inheritance

Severity: Info.

DataStore and MaintainerUtils:
DataStore is a simple storage pointer that allows contract storage to be isolated based on keys and IDs. This in effect creates arbitrary structs with an infinite number of parameters, which is good.

It is documented that MaintainerUtils provides a base class for some types like pools and operators. They inherit this arbitrary functionality to have maintainers. On top of this, the maintainer is a child class of DataStore; which, in effect, is extended by geodeUtils library to retain a base functionality of TYPE and CONTROLLER.

The structure is not well-defined and explicit. It is a safety risk to create arbitrary inheritances between unrelated libraries.

On top of this, no instance of the maintainerUtils can be seen in StakeUtils because this arbitrary inheritance is used to call the functions from MaintainerUtils, *with a DataStore pointer*. Which does not make any sense and makes things very difficult to review.

Mitigation:
Maintainer logic is revised, and MaintainerUtils library is removed. Some functions are moved into StakeUtils and most are deleted.

Arbitrary inheritance between maintainers, controllers, IDs, Pools, and Operators is removed from the codebase and the documentation.
Now, *maintainer* is not really different from another parameter, like *fee*. Can be utilized by the pools and operators to automate and optimize workflows, well protected and very limited.

Code is much easier to read and understand.

Post-Audit Improvements

Post Audit Improvements

1. Simplified Terminology

Contract	Former	Latter
gETH	_interfaceAvoiders	_avoiders
gETH	ERC1155Interfaces	gETHInterfaces
Swap	Dynamic Withdrawal Pools	Liquidity Pools
DataStoreUtils	DataStore	IsolatedStorage
DataStoreUtils	readUintForId, etc.	readUint, etc.
GeodeUtils → DataStoreUtils	getIdFromName	generateId
GeodeUtils	Universe	DualGovernance
GeodeUtils	Senate Elections	** removed **
GeodeUtils	_proposalForId	_proposals
GeodeUtils	GOVERNANCE_TAX	GOVERNANCE_FEE
StakeUtils	StakePool	PooledStaking
StakeUtils	Planet / Comet	Pool
StakeUtils	depositPlanet	deposit
StakeUtils	maintainerFee	maintenanceFee
StakeUtils	maintainerWallet	wallet
StakeUtils	getVersion	getContractVersion
WithdrawalContract	MiniGovernance	WithdrawalContract
WithdrawalContract	IsolationMode	recoveryMode

List of *notable* terminology changes is provided above. Additionally, related parameters, functions and folder names have changed respectively.

Geode Finance has a space theme. Previously, The Protocol has also followed this theme in its terminology. Smart Contracts and libraries were filled with arbitrary naming conventions, which don’t rely on any reasoning other than design choices.

Code readability and reliability has increased by removing all arbitrary naming conventions from the codebase.

Furthermore, a lot of functionality is removed from the contracts, and some contracts can contain different functionalities now.

2. Code Complexity, Sustainability and Readability

The reviewed version of The Protocol contained many non-trivial functionalities. These functionalities were either experimental or poorly executed. Many issues identified on external and internal audits were mostly originated from these functionalities.

Moreover, there were a lot of issues caused by the vague boundaries of libraries. Best code practices were not followed and interactions between libraries were unorganized. Most of them are mitigated now.

During the post-audit period, the most important commitments of the Team were:

1. Providing a secure implementation
2. Removing non-trivial or experimental methods
3. Improving the user experience
4. Improving the developer experience
5. Simplifying the codebase

To achieve these objectives, some precautions were taken:

1. Introduced globals.sol
2. Removed the MaintainerUtils library, revised the maintainer logic
3. Utilizes the DataStoreUtils view functions in Portal, instead of getters for arbitrary structs, such as getPlanet, getOperator.

4. Removed the Dynamic Withdrawals logic, and withdrawalBoost related effects.
5. Removing the 3-step validator withdrawal process: signal-oracle-fetch.
6. Simplifying the Initiators.
7. Simplifying the Oracle operations and separating it from operations related to StakeUtils.
8. Removing Senate Elections.
9. Not allowing the surplus to burn.
10. Converting approveOperators to batchApproveOperators

We have stated the limits of the abstract definitions explicitly, removed vague permissions and inheritance based understanding of the separated parties; like controllers and maintainers, planets and operators…

3. Improved Trustlessness and Security Assessment

The team has been working on The Protocol to ensure having the best trustlessness among all parties, and designed robust upgradability patterns and proper circuit breakers.

1. *updateStakingParams* is removed from Portal.
2. Circuit Breakers are implemented.
3. Maintainers are nerfed.
4. Senate can contain any functionality and is not limited to *votes* anymore.

1. Rogue Actors

a. What'd be the extent of a rogue Governance?

Governance can conduct a hostile takeover, **only when the Senate is expired.**

b. What'd be the extent of a rogue Senate?

Denial of service until it expires, this can take at most one year to resolve.

c. What'd be the extent of a rogue Pool Controller?

Nothing, Pool Controllers do not have access to the underlying collateral.

d. **What’d be the extent of a rogue Operator Controller?**

Malicious Operators can conduct griefing attacks causing stakers to lose principal.

e. **What’d be the extent of a rogue maintainer for Pools?**

Losing potential profit by choosing bad operators or refusing to choose operators.

f. **What’d be the extent of a rogue maintainer for Operators?**

Operators can get imprisoned as a result of the actions of a rogue maintainer. However, Governance can bail out the operators, if reasonable.

2. Hacks

a. **What’d be the extent if gETH is hacked and what are the circuit breakers in place?**

- Protocol-wide losses, no loss on the principal(staked ether).
- It would probably affect any protocol that integrated with gETH.
- gETH can be paused/unpaused **by Governance**.

b. **What’d be the extent if Portal is hacked and what are the circuit breakers in place?**

- Minimal losses.
- Only, the internal wallets of the operators and pool owners and the ether that is waiting to be staked can be harmed.
- Portal can be paused/unpaused **by Governance**.

c. **What’d be the extent if A Liquidity Pool is hacked and what are the circuit breakers in place?,**

- Up to 100% of the funds provided as liquidity can be lost.
- Liquidity Pools can be paused **by Governance**, for now.

d. **What’d be the extent if A withdrawal Contract is hacked and what are the circuit breakers in place?**

Varying level of losses:

- There is mostly almost no money kept in the withdrawal contracts, a small attack wouldn’t cause a lot of issues.
- But if the contract is hacked with a way to lock the funds, or prevent upgrades, then all funds related to the pool can be lost.
- Withdrawal Contracts can be paused **by Pool Controllers**.

Conclusion

File Name	Code Coverage				Reviews (pass)			Final (pass)
	Statements	Branches	Functions	Lines	Audit Result	Modification Degree	Internal Audit: Risk	
Portal.sol	90.48	76.47	86.36	90.91	🟥	🟨	🟨	🟨
gETH.sol	100.00	76.92	100.00	100.00	🟨	🟩	🟩	🟩
ERC20InterfacePermitUpgradable.sol	100.00	100.00	100.00	100.00	🟩	🟩	🟩	🟩
ERC20InterfaceUpgradable.sol	72.00	50.00	71.43	71.15	🟩	🟩	🟩	🟩
LPToken.sol	100.00	100.00	100.00	100.00	🟩	🟩	🟩	🟩
Swap.sol	100.00	100.00	100.00	100.00	🟩	🟩	🟩	🟩
AmplificationUtils.sol	100.00	100.00	100.00	100.00	🟩	🟩	🟩	🟩
MathUtils.sol	100.00	100.00	100.00	100.00	🟩	🟩	🟩	🟩
SwapUtils.sol	98.47	88.89	100.00	98.49	🟩	🟩	🟩	🟩
DataStoreUtilsLib.sol	100.00	100.00	100.00	100.00	🟩	🟨	🟩	🟩
DepositContractUtilsLib.sol	95.83	87.50	100.00	100.00	⬜	🟩	🟩	🟩
GeodeUtilsLib.sol	97.73	93.33	93.75	97.83	🟩	🟨	🟩	🟩
OracleUtilsLib.sol	100.00	100.00	100.00	100.00	🟥	🟥	🟨	🟥
StakeUtilsLib.sol	99.57	92.25	100.00	100.00	🟥	🟥	🟥	🟥
globals.sol	100.00	100.00	100.00	100.00	⬜	🟩	🟩	🟩
WithdrawalContract.sol	64.86	12.50	57.89	66.67	🟥	🟥	🟥	🟥

Conclusion

The Team is exceedingly satisfied with the review conducted by the Consensys Diligence auditors.

The Protocol is very suitable for future improvements that won't require a lot of changes to the existing contracts. Upgradability patterns exercised within The Protocol were designed with this very idea in mind. Considering the modular architecture of The Protocol, and the customizability of its products, it is obvious that there will be many audit necessities in the future.

Ideally, the Team is keen to continue working with Consensys Diligence for any future audits and establish an internal partnership.

Thus, the Team decided to move forward with Consensys Diligence for a follow-up audit that will be conducted prior to the launch of The Protocol on Ethereum Mainnet.

While the details are not finalized, a follow-up audit will start on April 10, 2023.

However, possibly the Team can not afford a comprehensive follow-up audit that will cover all of the changes explained in this document, due to financial constraints. Additionally, due to Shanghai Upgrade being scheduled for 12th April, it is crucial that The Protocol is launched as soon as possible...

Most of the modifications can be classified as simple improvements, refactoring of the existing code, and functional deductions which are not effectively breaking. Collectively, these modifications helped create smart contracts that work smoother, are easier to review, and more secure. Finally, the test coverage is boosted up to nearly 100% on many smart contracts and the Team will continue improving it before the mentioned Mainnet launch.

With the above reasoning, the Team conducted an internal review to identify the risk profile of the smart contracts. The internal review ended as of March 16, 2023. Many small issues were detected and mitigated, the codebase was finalized. In conclusion, the review identified 2 smart contracts that require a follow-up audit:

1. **StakeUtils.sol**, which contains functionality for The Staking Library.
2. **OracleUtils.sol**, which contains functionality for the Oracle operations.

Finally, a comprehensive internal audit is being conducted and will be finalized before the 10th of April, the start date of the follow-up audit. The team expects the audit to start with more additional documentation and at most some minor changes.

The Team hereby confirms that all of the changes that need to be reviewed were documented in this report.

StakeUtils.sol
https://github.com/Geodefi/Portal-Eth/blob/6f6ef6c701f0a6bfe4a190fd1399bca334b96ef4/contracts/Portal/utils/StakeUtilsLib.sol

OracleUtils.sol
https://github.com/Geodefi/Portal-Eth/blob/6f6ef6c701f0a6bfe4a190fd1399bca334b96ef4/contracts/Portal/utils/OracleUtilsLib.sol

Commit hash
https://github.com/Geodefi/Portal-Eth/tree/6f6ef6c701f0a6bfe4a190fd1399bca334b96ef4