GeodeFinance

Internal Audit Report

March 2023

Prepared by Crash Bandicoot
Geodefi Ltd.

1. Executive Summary

This report shows the findings of the internal audit identified while reviewing the Geode Finance's code base.

The audit was executed over 4 weeks, from March 6th, 2023 to March 31th, 2023, by Crash Bandicoot. A total of 20 person-days were spent. During this period, discussions regarding the findings and mitigations were held with Ice Bear.

Throughout the audit, Geode Finance's staking solution was reviewed, named The Staking Library. Implementation aims to provide a global standard for Permissionless Configurable Staking Pools and Derivatives. Geode Finance's goal is to create a staking library that can be utilized by a router, called Portal, by any parties easily. The Protocol hopes to contribute to the staking ecosystem with a trustless and decentralized solution.

Compared to the previous audit conducted by Diligence, code is simplified a lot and some naming conventions are corrected. However, there is more to cover, over the solidity style guideline. An explanation can be found in the Code Quality Appendix.

Staking logic in the provided contracts are working together with an Oracle logic. The functionality and security is highly dependent on the off-chain infrastructure and it is highly recommended to have an external audit on it.

The Protocol lacks the Withdrawal logic. However, since all other logics are implemented while keeping the withdrawal logic in mind; there are no other changes planned in any library or contract, other than the Withdrawal contract. Though, it still causes some ambiguity while reviewing the codebase.

Written tests are found enough but there is still some room for improvements like fuzz testing. Also, currently all tests are integration tests, which pass through Portal. They may not reach all the parts of the library calls. It is also recommended to write unit tests.

2. Scope

The review focused on the commit hash dafdbabbdfc9807e0e697cfec5a01884a9fef573. The list of files in scope can be found in the Appendix. Since some findings are fixed during the internal audit, some new findings may be related to the current version of the contracts.

2.1. Objectives

With Ice Bear, the following objectives were identified:

1. Changes from the previous audit are fixed correctly and not causing any other unintended consequences.
2. Report any known vulnerabilities in smart contract and DeFi related systems.
3. Figuring out problematic edge cases.

3. System Overview

For the detailed system overview, please refer to documentation.

4. Recommendations

4.1. External Audits

Changes from the Previous Audit

Although there is an internal audit conducted, it is a best practice to process all changed contracts with an external auditor, for an extra eye.

Withdrawal Contract

After the withdrawal contract is finalized, it needs to be externally audited.

Off-chain Infrastructure

As mentioned previously, off-chain infrastructure needs to be audited since the project's functionality and security is highly dependent on it. Price Oracle

Manipulation is one of the most common attacks with reentrancy in the DeFi ecosystem.

4.2. Review the Code Quality recommendations in Appendix

Some other comments related to code quality can be found under Appendix section Code Quality Recommendations.

5. Findings

Each issue has an assigned severity:

- **Gas** issues are recommendations for improvements on gas cost. Not required to be addressed, but it will make projects cheaper to use if code maintainers decide to address them.
- **Minor** issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- **Medium** issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- **Major** issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- **Critical** issues are directly exploitable security vulnerabilities that need to be fixed.

5.1. Pushing id two times to the list *allIdsByType*

Critical

Planet Ids pushed to allIdsByType list both at *approveProposal* and while initialization of the pool. This creates a problem while adding this type as an elector, since using length of the inner list that holds specific type's ids, it will mess with the elector count since some ids actually appear twice so it will mess with the elections.

```
307   function approveProposal(
308     DualGovernance storage self,
309     DSU.IsolatedStorage storage DATASTORE,
310     uint256 id
311   ) external onlySenate(self) returns (uint256 _type, address _controller) {
312     require(
313       self._proposals[id].deadline > block.timestamp,
314       "GU: NOT an active proposal"
315     );
316
317     _type = self._proposals[id].TYPE;
318     _controller = self._proposals[id].CONTROLLER;
319
320     require(_type != ID_TYPE.SENATE, "GU: can NOT approve SENATE election");
321
322     DATASTORE.writeUintForId(id, "TYPE", _type);
323     DATASTORE.writeAddressForId(id, "CONTROLLER", _controller);
324     DATASTORE.writeBytesForId(id, "NAME", self._proposals[id].NAME);
325     DATASTORE.allIdsByType[_type].push(id);
```

contracts/Portal/utils/GeodeUtilsLib.sol: L325

```
395   function setElectorType(
396     DualGovernance storage self,
397     DSU.IsolatedStorage storage DATASTORE,
398     uint256 _TYPE,
399     bool _isElector
400   ) external onlyGovernance(self) {
401     require(_isElector != isElector(self, _TYPE), "GU: type already elector");
402     require(
403       _TYPE > ID_TYPE.__GAP__,
404       "GU: 0, Senate, Upgrade, GAP cannot be elector"
405     );
406
407     self._electorTypes[_TYPE] = _isElector;
408
409     if (_isElector) {
410       self._electorCount += DATASTORE.allIdsByType[_TYPE].length;
411     } else {
412       self._electorCount -= DATASTORE.allIdsByType[_TYPE].length;
413     }
```

contracts/Portal/utils/GeodeUtilsLib.sol: L410

```
552   function initiatePool(
553     PooledStaking storage self,
554     DSU.IsolatedStorage storage DATASTORE,
555     uint256 fee,
556     uint256 interfaceVersion,
557     address maintainer,
558     address _GOVERNANCE,
559     bytes calldata NAME,
560     bytes calldata interface_data,
561     bool[3] calldata config
562   ) external {
563     require(
564       msg.value == DCU.DEPOSIT_AMOUNT,
565       "SU: requires 1 validator worth of Ether"
566     );
567
568     uint256 id = DSU.generateId(NAME, ID_TYPE.POOL);
569
570     require(id > 10 ** 7, "SU: Wow! low id");
571
572     require(
573       DATASTORE.readUintForId(id, "initiated") == 0,
574       "SU: already initiated"
575     );
576     DATASTORE.writeUintForId(id, "TYPE", ID_TYPE.POOL);
577     DATASTORE.writeAddressForId(id, "CONTROLLER", msg.sender);
578     DATASTORE.writeBytesForId(id, "NAME", NAME);
579     DATASTORE.writeUintForId(id, "initiated", block.timestamp);
580     DATASTORE.allIdsByType[ID_TYPE.POOL].push(id);
581
```

contracts/Portal/utils/StakeUtilsLib.sol: L580

Recommendation:

Since pools are permissionless, add a check in the newProposal function to not accept pool proposals. It will solve this issue.

Status:

In the newProposal of GoedeUtils library, we now prevent creating a proposal for a Pool (type 5). So, this issue is Already addressed and fixed.

5.2. *isPriceValid* check is faulty

Critical

Expected behavior of the price validity is to be invalid for all the Pools after Oracle reports new price merkle root. Since in the require statement, or (||) is used instead of and (&&) it lets Pools use the previous price up to 24 hours (PRICE_EXPIRY) more with the previous price.

```
1080    function isPriceValid(
1081      PooledStaking storage self,
1082      uint256 poolId
1083    ) public view returns (bool isValid) {
1084      uint256 lastupdate = self.gETH.priceUpdateTimestamp(poolId);
1085      unchecked {
1086        isValid =
1087          lastupdate + PRICE_EXPIRY >= block.timestamp ||
1088          lastupdate >= self.ORACLE_UPDATE_TIMESTAMP;
1089      }
1090    }
```

contracts/Portal/utils/StakeUtilsLib.sol: L1087

Recommendation:

Or (||) statement needs to be changed with the and (&&) statement.

Status:

Already addressed and fixed.

5.3. *priceSync* has not check for redo to increase *priceValidity*

Critical

priceSync function has no checks for re-doing the price update even while already done with the current price merkle root. This lets Pool to increase its validity by updating Pool's price update timestamp. And with the previous issue 5.2, it lets Pool to use the price and continue doing deposits even though oracle is not giving any price updates to the infinite.

Recommendation:

Add a *require* statement to check if the Pool's price update timestamp is smaller than oracles update timestamp to do the _priceSync

Status:

Already addressed and fixed.

5.4. Non-existent type can be proposed, contract manipulation risk for later

Major

In the *newProposal*, check is needed to be done to be sure the given type actually exists. Also same for *setElectorType* function, it only checks if the type is bigger than a constant number. Not existing types can be given and creates a risk for contracts currently or with the later updates since those typed Ids will keep staying there to the infinite.

Recommendation:

An "existingTypes" mapping can be created to keep being modular. And can add new types to that mapping with a proposal and check the type exists in that mapping in the necessary functions.

Status:

Since *setElectorType* is deleted this is no longer a problem.

5.5. Senate election manipulation is possible with *setElectorType*

Major

To not affect the senate elections, during the senate proposal, there should not be any change at election types (*setElectorType* should not be functional during the elections) and it is also better not to accept any proposal, at least from an elector type.

```
395   function setElectorType(
396     DualGovernance storage self,
397     DSU.IsolatedStorage storage DATASTORE,
398     uint256 _TYPE,
399     bool _isElector
400   ) external onlyGovernance(self) {
401     require(_isElector != isElector(self, _TYPE), "GU: type already elector");
402     require(
403       _TYPE > ID_TYPE.__GAP__,
404       "GU: 0, Senate, Upgrade, GAP cannot be elector"
405     );
406
407     self._electorTypes[_TYPE] = _isElector;
408
409     if (_isElector) {
410       self._electorCount += DATASTORE.allIdsByType[_TYPE].length;
411     } else {
412       self._electorCount -= DATASTORE.allIdsByType[_TYPE].length;
413     }
414
415     emit ElectorTypeSet(_TYPE, _isElector);
416   }
```

contracts/Portal/utils/GeodeUtilsLib.sol: L395 - L416

Recommendation:

Add require statement to prevent *setElectorType* function to be called during the senate election.

Status:

Since setElectorType and senate logic is deleted. Issue is resolved.

5.6. *changeMaintainer* can be griefed by a malicious maintainer

Major

changeMaintainer is only allowed if not prisoned for the operator but the controller could lose the control of the maintainer, like a malicious maintainer. And malicious maintainers can keep being malicious and do not let controllers to change it causing the abolish the use of changeMaintainer and so other maintainer controlled functions. So this logic needs to be reconsidered.

Recommendation:

changeMaintainer function should not use authenticate, do the checks there directly.

Status:

Already addressed and fixed.

5.7. *authenticate* function faulty call causes some functions being out of use

Medium

authenticate function is problematic, both “expectedMaintainer” and “expectedCONTROLLER” parameters are set as true for some calls, but for it to be true, controller and maintainer should be the same. This is not correct for all cases. This problem can be found in the following functions: *batchApproveOperators*, *proposeStake*, *beaconStake and switchValidatorPeriod*.

```
223   function authenticate(
224     DSU.IsolatedStorage storage DATASTORE,
225     uint256 id,
226     bool expectCONTROLLER,
227     bool expectMaintainer,
228     bool[2] memory restrictionMap
229   ) internal view {
230     require(
231       DATASTORE.readUintForId(id, "initiated") != 0,
232       "SU: ID is not initiated"
233     );
234
235     uint256 typeOfId = DATASTORE.readUintForId(id, "TYPE");
236
237     if (typeOfId == ID_TYPE.OPERATOR) {
238       require(restrictionMap[0], "SU: TYPE NOT allowed");
239       if (expectCONTROLLER || expectMaintainer) {
240         require(
241           !isPrisoned(DATASTORE, id),
242           "SU: operator is in prison, get in touch with governance"
243         );
244       }
245     } else if (typeOfId == ID_TYPE.POOL) {
246       require(restrictionMap[1], "SU: TYPE NOT allowed");
247     } else revert("SU: invalid TYPE");
248
249     if (expectMaintainer) {
250       require(
251         msg.sender == DATASTORE.readAddressForId(id, "maintainer"),
252         "SU: sender NOT maintainer"
253       );
254       return;
255     }
256
257     if (expectCONTROLLER) {
258       require(
259         msg.sender == DATASTORE.readAddressForId(id, "CONTROLLER"),
260         "SU: sender NOT CONTROLLER"
261       );
262       return;
263     }
264   }
```

contracts/Portal/utils/StakeUtilsLib.sol: L223 - L264

```
986     function batchApproveOperators(
987       DSU.IsolatedStorage storage DATASTORE,
988       uint256 poolId,
989       uint256[] calldata operatorIds,
990       uint256[] calldata allowances
991     ) external returns (bool) {
992       authenticate(DATASTORE, poolId, true, true, [false, true]);
```

contracts/Portal/utils/StakeUtilsLib.sol: L992

```
1318    function proposeStake(
1319      PooledStaking storage self,
1320      DSU.IsolatedStorage storage DATASTORE,
1321      uint256 poolId,
1322      uint256 operatorId,
1323      bytes[] calldata pubkeys,
1324      bytes[] calldata signatures1,
1325      bytes[] calldata signatures31
1326    ) external {
1327      // checks and effects
1328      authenticate(DATASTORE, operatorId, true, true, [true, false]);
```

contracts/Portal/utils/StakeUtilsLib.sol: L1328

```
1465    function beaconStake(
1466      PooledStaking storage self,
1467      DSU.IsolatedStorage storage DATASTORE,
1468      uint256 operatorId,
1469      bytes[] calldata pubkeys
1470    ) external {
1471      authenticate(DATASTORE, operatorId, true, true, [true, false]);
```

contracts/Portal/utils/StakeUtilsLib.sol: L1471

Recommendation:

Only give access either to controller or maintainer for these functions.

Status:

Already addressed and fixed.

5.8. Withdrawal contract upgrade proposal while Pool is not ready yet

Medium

fetchWithdrawalContractUpgradeProposal function does not have any restrictions, anyone can call it this may lead a Pool to make a decision without even deciding how to proceed yet.

```
688  function fetchWithdrawalContractUpgradeProposal(
689    uint256 id
690  ) external virtual override returns (uint256 withdrawalContractVersion) {
691    withdrawalContractVersion = STAKER._defaultModules[
692      ID_TYPE.MODULE_WITHDRAWAL_CONTRACT
693    ];
694
695    StakeUtils.withdrawalContractById(DATASTORE, id).newProposal(
696      DATASTORE.readAddressForId(withdrawalContractVersion, "CONTROLLER"),
697      ID_TYPE.CONTRACT_UPGRADE,
698      DATASTORE.readBytesForId(withdrawalContractVersion, "NAME"),
699      4 weeks
700    );
701  }
```

contracts/Portal/Portal.sol: L688 - L701

Recommendation:

Make sure not everyone can call this function as they wish with any Pool Id.

Status:

Already addressed and fixed.

5.9. Missing zero checks in Withdrawal Contract

Medium

Missing zero checkers in withdrawal contract *initialize* function.

```
62  function initialize(
63    uint256 _VERSION,
64    uint256 _ID,
65    address _gETH,
66    address _PORTAL,
67    address _OWNER
68  ) public virtual override initializer returns (bool) {
69    __ReentrancyGuard_init();
70    __Pausable_init();
71    __UUPSUpgradeable_init();
72
73    gETH = _gETH;
74    POOL_ID = _ID;
75
76    GEM.GOVERNANCE = _PORTAL;
77    GEM.SENATE = _OWNER;
78    GEM.SENATE_EXPIRY = type(uint256).max;
79
80    CONTRACT_VERSION = _VERSION;
81    emit ContractVersionSet(_VERSION);
82
83    return true;
84  }
```

contracts/Portal/liquidityPool/utils/SwapUtils.sol: L91

Recommendation:

Check that addresses are not zero.

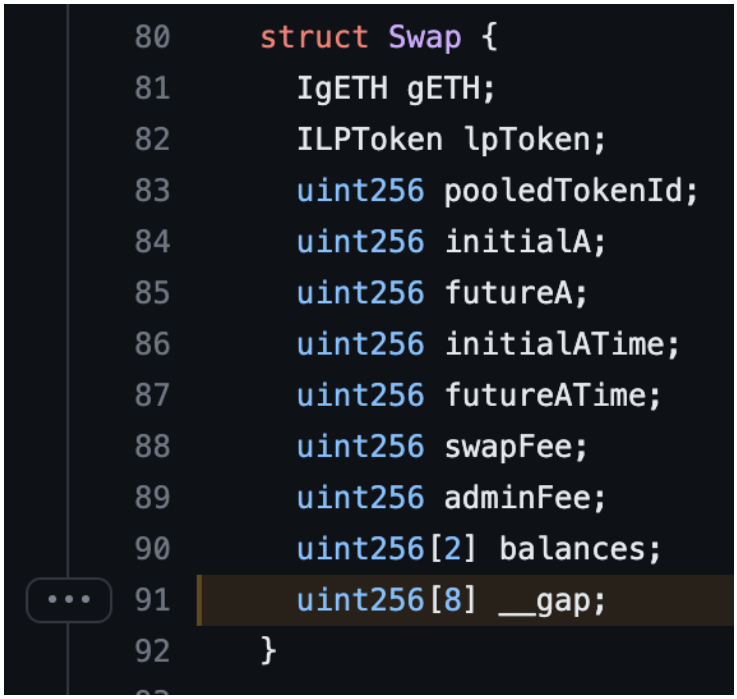
Status:

Already addressed and fixed.

5.10. Gap is faulty in the Swap struct

Medium

Swap struct's gap is wrong. It is a uint256[8] __gap but it should be 5.



```
80    struct Swap {
81      IgETH gETH;
82      ILPToken lpToken;
83      uint256 pooledTokenId;
84      uint256 initialA;
85      uint256 futureA;
86      uint256 initialATime;
87      uint256 futureATime;
88      uint256 swapFee;
89      uint256 adminFee;
90      uint256[2] balances;
91      uint256[8] __gap;
92    }
```

contracts/Portal/liquidityPool/utils/SwapUtils.sol: L91

Recommendation:

Change the 8 with 5.

Status:

Already addressed and fixed. Also checked all gap identifiers of structs and contracts.

5.11. Monopoly threshold is set to max on initialization

Minor

Monopoly threshold is set to max at the beginning, it causes one to create a Pool and pass the threshold of the ethereum in the first few hours of the protocol until the first oracle update comes. Practically it may seem impossible, but this is possible like if Lido or Rocketpool comes during the initiation of the protocol and they can put all their validator and pass the threshold easily. They can stay upon the threshold until they exit.

```
199        STAKER.MONOPOLY_THRESHOLD = type(uint256).max;
```

contracts/Portal/Portal.sol: L199

Recommendation:

Set it to min until the first oracle report comes or something less than the planned threshold.

Status:

Acknowledged but not addressed as there is no likelihood of this happening within the first 8 hours of the launch.

5.12. 32 ETH to initiate a Pool won't make sense if it can be withdrawn

Minor

While writing a withdrawal contract one should be careful and should not add a direct way out for gETH. Otherwise malicious parts can get money from lending and handle it in one transaction which will not provide the wanted restriction.

Recommendation:

Can put it directly to the surplus and lock it for some time or until an operator takes it.

Status:

It is stated that all funds that are entering the pool need to cycle through the Beacon chain, before ending up on the Withdrawal Contract.

5.13. Uninitialized variables

Minor

Some variables are not initialized, even though they are zero, it is a good practice to initialize it as 0. *appendAddressArrayBatch*, *regulateOperators*, *stakeBeacon*, loop parameters("i" or "j") are not initialized. Also "lastIdChange" in *stakeBeacon*.

Recommendation:

Initialize those parameters as the correct initial value.

Status:

Already addressed and fixed for looping variables (i,j) some local variables can stay.

5.14. Unused return statements

Minor

deposit and *deposit,* returns bought and minted gETH but return not used, statement. Same occurs for *batchApproveOperators* and *approveOperators,* it returns a bool statement but never used.

Recommendation:

Either use the return or delete the return from the called functions.

Status:

Already addressed and fixed. Using the return statements now.

5.15. governanceFee already must be smaller than Max

Minor

Checking the fee cooldown at *getGovernanceFee* will cost much more gas, instead it can be checked while setting the fee at *setGovernanceFee* which will be much more gas efficient.

Also since the MAX_GOVERNANCE_FEE is constant from now on, no need to check for. If MAX_GOVERNANCE_FEE > self.GOVERNANCE_FEE, because it is impossible to set a fee bigger than MAX_GOVERNANCE_FEE.

```
 99    /**
100     * @notice limiting the GOVERNANCE_FEE, 5%
101     */
102    uint256 public constant MAX_GOVERNANCE_FEE =
103      (PERCENTAGE_DENOMINATOR * 5) / 100;
```

contracts/Portal/utils/GeodeUtilsLib.sol: L102

```
176     function getGovernanceFee(
177       DualGovernance storage self
178     ) external view returns (uint256) {
179       return
180         block.timestamp < FEE_COOLDOWN
181           ? 0
182           : MAX_GOVERNANCE_FEE > self.GOVERNANCE_FEE
183           ? self.GOVERNANCE_FEE
184           : MAX_GOVERNANCE_FEE;
185     }
```

contracts/Portal/utils/GeodeUtilsLib.sol: L182

```
191     /**
192      * @notice onlyGovernance, sets the governance fee
193      * @dev Can not set the fee more than MAX_GOVERNANCE_FEE
194      */
195     function setGovernanceFee(
196       DualGovernance storage self,
197       uint256 newFee
198     ) external onlyGovernance(self) {
199       require(newFee <= MAX_GOVERNANCE_FEE, "GU: > MAX_GOVERNANCE_FEE");
200
201       self.GOVERNANCE_FEE = newFee;
202
203       emit GovernanceFeeUpdated(newFee);
204     }
```

contracts/Portal/utils/GeodeUtilsLib.sol: L195 - L204

Recommendation:

Delete the check MAX_GOVERNANCE_FEE > self.GOVERNANCE_FEE

Status:

Not addressed, Icebear did not accept this as an issue stating that getter functions should always enforce limits.

5.16. Not resetting the whitelist can cause unintended behavior

Minor

While setting a private pool as public with a whitelisting contract, will eliminate the whitelisting contract. But when it is set as private again, even though not intended, it will keep the previous whitelisting contract. Which should not be the expected behavior.

Recommendation:

Set the whitelisting contract as zero address for the Pool while setting it to public.

Status:

Already addressed and fixed.

5.17. Gas - All gas findings:

- GeodeUtils using the ***isElector*** function inside the ***setElectorType*** function and also in the ***approveSenate*** function. Instead, one can just use the code itself to save gas.

- DataStoreUtils (usage in GeodeUtils and maybe somewhere else too) can have unchecked versions of ***addUintForId*** and similar functions. Since there are parts that can be sure, it is not possible to overflow. Like voting for the senate in the ***approveSenate*** function.

- There is no need to set type to *“memory uint256 _type = self._proposals[proposalId].TYPE;”* in GeodeUtils, since it is only used once in the function ***approveSenate***

- To save gas, “*WC:*” can be used with the require error messages instead of “*WithdrawalContract:*

- **Custom errors** can be used instead of require statements, to save gas!

- OracleUtils ***onlyOracle*** modifier, gets the whole storage struct as a parameter, just getting the address saves gas. Tested!

- Assignment *“uint256 operatorId = STAKER._validators[_pk].operatorId;”* definitely needs to be taken front since it is used one more time beforehand. So it will save gas!

- In function ***approveProposal*** in Portal, instead of checking all default types one by one, a mapping can be created from type number ⇒ “default” or “allowed” strings or an integer again like 1 and 2 for “default” and “allowed” respectively to save gas, this will also combined with recommendation in **issue 5.4**.

- StakeUtils ***_setMaintainer*** function, unnecessary memory variable is used, no need to assign the “currentMaintainer” to a

variable, can be used in the require statement directly to save gas.

- StakeUtils *"SU: not enough funds in Portal ?"* unnecessary empty space before question mark, delete it to save gas.

- StakeUtils ***authenticate*** function is a gas monster. Each check can be written one by one to save gas. As an example, in the ***batchApproveOperators*** function, let's say one sets 10 operators. For each operator, in the for loop it is doing 3 unnecessary if statement checks and an unnecessary require statement related with the maintainer and controller; which cause 180 more gas to be consumed then needed.

- About loops which have checks in it, can do the checks first in another previous loop first. Then do the statements in another loop, because if the required case fails in the last iteration, the user will **lose all the previous gas**. For instance, this makes sense at the proposeStake function's loop and ***batchApproveOperators*** function's loop too.

- No need for the decreased parameter in ***_decreaseWalletBalance*** function. Since it will fail and revert anyways. Otherwise it will always return true. There is no case you return false. Then it makes no sense to return true.

- Instead of **feetheft** and **alienate** events, they both call ***_imprison***. ***_imprison*** can get another parameter for "calledType" which is *bytes*. For **feetheft** the parameter can be

the event name and for the **alienate** it could be the *PK*. Need to fix any other place ***_imprison*** is called if there is.

- Geodeutils *" GU: already approved",* there is an extra space at the beginning.

APPENDIX

1. Code Quality Recommendations

1.1. Style

- State variables should take place before events according to the solidity style guide.

- Function order should be changed according to the solidity style guide function order.

- Geodeutils ***getGovernanceFee*** function is no need to use ***ternary***, to make if more simple use normal if statements, if won't give you optimization in any ways, writing code more clear will give you less possibility to make errors and readability by auditors.

- What is the need of _ (underscore) before some variables inside the struct? There is no need if there is no name collision and there is not.

- WithdrawalContract → no need for the word contract in the name, since it is already a contract.

- Everywhere (mostly structs) using all capital variable names. This makes one think that these are constants, while they are not. Which is a bad practice for the style guide.

- Delete all todo's before going production "todo: add to portal isWhiteListed", not added to portal, check it again and add it if not there!

- StakeUtils ***withdrawalContractById*** and ***liquidityPoolById*** functions taking the *"poolId"* parameter differently, and having a naming convention is needed.

- In StakeUtils, put **constantValidatorData** Struct to the beginning of the library.

- StakeUtils **proposeStake** function, in the **Validator** Struct creation; for the state, do not use 1, instead use VALIDATOR_STATE.PROPOSED

- In GeodeUtils, 1743454800 timestamp is given as feeless time, giving a timestamp is confusing, instead use reserved words like 2 years, 1 years.

1.2. Logic

- In GeodeUtils, `*self._proposals[proposalId].deadline >= block.timestamp, "GU: proposal expired"*` is using *">="*, all other checks related to this using *">"*, for consistency, it is good to use *">"*.

- In GeodeUtils, instead of the *" GU: already approved"* message, *“GU: already voted”* is a much less confusing message.

- In OracleUtils *"OU: NOT all pubkeys are pending"* message, change pubkeys with pubkey. That way makes more sense, since there is only one pubkey in that function ***_alienateValidator***.

- OracleUtils *"OU: NOT all proofs are valid"* message is better to be changed with *“OU: proof is NOT valid”* message, since not only the batch function is calling and it is a single proof not valid at that moment.

- StakeUtils ***_deployInterface*** function’s require statement *"SU: could not init interface"* cannot be triggered. Because either it will revert during the initialization or return true, it will never return false so this requirement is basically unnecessary. try catch can be used for that external call instead of require, and use revert inside the catch block.

- Same problem with the previous Appendix **1.2.6** ***deployLiquidityPool*** ISwap ***initialization***.

- StakeUtils needs an event to detect the public and private pools when changed.

- In GeodeUtils, *`require(_isElector != isElector(self, _TYPE), "GU: type already elector")`* statement's error message is not accurate since it may also *"already be not an elector"*.

1.3. Improvement

- Withdrawal contract is required to deploy in the current protocol; but for a solo staker, it is not needed to be. It can be made configurable during the initialization. if not creating a pool and not having an interface and so on, can get the given WC address as EOA. Not giving any ability to open a pool or creating an interface later on to those controllers. This way the protocol can let them use the system as it is without even needing to deploy a withdrawal contract and give gas for that or try to keep it updated for every withdrawal contract update.

- Let controllers of the liquidity pools' have a pause functionality for acting fast when protocol starts using the real governance!

2. Files in Scope

Internal audit was conducted on the following files:

File	Link
code/contracts/Portal/gETH.sol	here
code/contracts/Portal/Portal.sol	here
code/contracts/Portal/utils/globals.sol	here
code/contracts/Portal/utils/DataStoreUtilsLib.sol	here
code/contracts/Portal/utils/GeodeUtilsLib.sol	here
code/contracts/Portal/utils/OracleUtilsLib.sol	here
code/contracts/Portal/utils/StakeUtilsLib.sol	here
code/contracts/Portal/withdrawalContract/withdrawalContract.sol	here

3. Usage

This report belongs to GEODE FINANCE and it is not allowed to share with other parties. Even though GEODE FINANCE shares this report with you or with your company, it should not be shared with any other parties.