our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Geode Finance

Withdrawal Module

SECURITY REVIEW

Date: 6 October 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.

Learn more about Geode's concept and the technicalities behind it here.

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.

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible
- **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 20 days and a total of 480 hours have been spent by the three auditors:

- @ShieldifyMartin
- @ShieldifyAnon
- @ShieldifyGhost

The project has undergone several audits prior to this one (both external and internal, one of the external by Shieldify Security). However, this is the first audit for Geode's Withdrawal module, which is at the center of this audit's scope.

The Withdrawal module oversees withdrawal requests and exit elections. It also handles tasks such as processing withdrawals, distributing fees, withdrawal request queuing, and conducting exit elections through an instant `run-off` method, all while upholding validator segregation.

Some of the more noteworthy issues that the audit identified cover the possibility for a user to have his funds stucked and a proposal for improved safety when transferring the ownership of a withdraw request. There are also a few Informational findings that include tips for the implementation of good practices and a small set of gas optimization techniques that would ultimately improve the UX, in the context of execution fees.

This specific part of the Geode protocol is still not included in the project's documentation. Nevertheless, considering the tight timeframe, the developers have done outstanding job with describing the module's mechanics via a comprehensive natspec. The added examples in the codebase's comments further aid the understanding of the logic.

We would also like to thank the Geode team for being very responsive and for providing clarifications and detailed responses to all of our questions. They are an amazing project and Shieldify is happy to be part of it.

5.1 Protocol Summary

Project Name	**Geode Finance**
Repository	Portal-Eth
Type of Project	Decentralized and Liquid Staking Pools
Audit Timeline	20 days
Review Commit Hash	8b5c94317bd2b09567cb6ed00d50305b9d923bd0
Fixes Review Commit Hash	4c59a1214ac6d2e84b0fe5d020f00fdb306ffcc5

5.2 Scope

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

File	nSLOC
contracts/Portal/packages/WithdrawalContract.sol	80
contracts/Portal/modules/WithdrawalModule/WithdrawalModule.sol	97
contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol	294
contracts/Portal/middlewares/ERC20RebaseMiddleware.sol	124
Total	**595**

6. Findings Summary

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

- **Critical** and **High** issues: **4**
- **Medium** issues: **0**
- **Low** issues: **2**
- **Informational** issues: **8**
- **Gas Optimization** issues: **7**

ID	Title	Severity
[C-01]	Calculation Mistake on `_fulfill()` And `_fulfillBatch()` Functions	Critical
[C-02]	Queue `fulfilled` Parameter Should be Increased on a `fulfill` Operation	Critical
[H-01]	User's Staked ETH Can Be Stuck in the Protocol Through Two Cases	High
[H-02]	`processValidators()` Function Can Be Prevented Funds Get Stuck	High
[L-01]	`Re-issued` Votes Are Vulnerable	Low
[L-02]	Withdrawal Request Could be Assigned a Wrong New Owner	Low
[I-01]	Misleading `require` Message	Informational
[I-02]	Missing Event Emission in `processValidators()`	Informational
[I-03]	Missing Check for the New Variable in `setExitThreshold`	Informational
[I-04]	Remove Unused Variables	Informational
[I-05]	Use Scientific Notation Rather Than Exponentiation	Informational
[I-06]	Variables Naming does not Follow the Solidity Style Guide	Informational
[I-07]	Open TODOs	Informational
[I-08]	Typos	Informational
[G-01]	No Need to Initialize Variables with Default Values	Gas Optimization
[G-02]	Use `calldata` Instead of `memory` for Function Arguments that do not get Mutated	Gas Optimization
[G-03]	Cache Array Length Outside of Loops	Gas Optimization
[G-04]	Revert Strings Can be Shortened	Gas Optimization
[G-05]	`++i` Costs Less Gas Than `i++`	Gas Optimization
[G-06]	Use `!= 0` Instead of `> 0` for Unsigned Integer Comparison	Gas Optimization
[G-07]	Do not Calculate Constants	Gas Optimization

7. Findings

[C-01] Calculation Mistake on `_fulfill()` And `_fulfillBatch()` Functions

Severity

Critical Risk

Description

On line 510 should consider using `gETH.denominator()` when increasing the claimableETH on `_fulfill()`.

On line 486 should consider using `gETH.denominator()` increasing the claimableETH on `_fulfillBatch()`.

This finding has been identified in close collaboration with the Geode core developer team.

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol

```
function _fulfill(PooledWithdrawal storage self, uint256 index) internal
    {
  uint256 toFulfill = fulfillable(self, index, self.queue.realized, self.
     queue.fulfilled);
  if (toFulfill > 0) {
    self.requests[index].claimableETH += toFulfill * self.queue.
       realizedPrice;
    self.requests[index].fulfilled += toFulfill;
    self.queue.fulfilled = toFulfill;
  }
}

function _fulfillBatch(
  PooledWithdrawal storage self,
  uint256[] calldata indexes,
  uint256 Qrealized,
  uint256 Qfulfilled,
  uint256 Qprice
) internal {
  for (uint256 i = 0; i < indexes.length; i++) {
    uint256 toFulfill = fulfillable(self, indexes[i], Qrealized,
       Qfulfilled);
    if (toFulfill > 0) {
      self.requests[indexes[i]].claimableETH += toFulfill * Qprice;
      self.requests[indexes[i]].fulfilled += toFulfill;
      Qfulfilled += toFulfill;
    }
  }
  self.queue.fulfilled = Qfulfilled;
}
```

Recommendation

Add the `denominator()` function when increasing `claimableETH` request.

```
function _fulfill(PooledWithdrawal storage self, uint256 index) internal
    {
  uint256 toFulfill = fulfillable(self, index, self.queue.realized, self.
     queue.fulfilled);
  if (toFulfill > 0) {
-    self.requests[index].claimableETH += toFulfill * self.queue.
    realizedPrice;
+    self.requests[index].claimableETH += (toFulfill * self.queue.
    realizedPrice) / self.gETH.denominator();
     self.requests[index].fulfilled += toFulfill;
     self.queue.fulfilled = toFulfill;
     self.queue.fulfilled += toFulfill;
  }
}

function _fulfillBatch(
  PooledWithdrawal storage self,
  uint256[] calldata indexes,
  uint256 Qrealized,
  uint256 Qfulfilled,
  uint256 Qprice
) internal {
  for (uint256 i = 0; i < indexes.length; i++) {
    uint256 toFulfill = fulfillable(self, indexes[i], Qrealized,
       Qfulfilled);
    if (toFulfill > 0) {
-      self.requests[indexes[i]].claimableETH += toFulfill * Qprice;
+      self.requests[indexes[i]].claimableETH += (toFulfill * Qprice) /
    self.gETH.denominator();
       self.requests[indexes[i]].fulfilled += toFulfill;
       Qfulfilled += toFulfill;
    }
  }
  self.queue.fulfilled = Qfulfilled;
}
```

Team Response

Acknowledged, will be mitigated.

[C-02] Queue `fulfilled` Parameter Should be Increased on a `fulfill` Operation

Severity

Critical Risk

Description

On line 490 of the **`_fulfill()`** function, **`self.queue.fulfilled`** is set to **`toFulfill`**. However, it should have been an **`addition`**, because we actually want to **`increase`** the **`fulfilled`** part after a **`fulfill`** call.

This finding has been identified in close collaboration with the Geode core developer team.

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol#L486

```
function _fulfill(PooledWithdrawal storage self, uint256 index) internal
    {
  uint256 toFulfill = fulfillable(self, index, self.queue.realized, self.
     queue.fulfilled);
  if (toFulfill > 0) {
    self.requests[index].claimableETH += toFulfill * self.queue.
       realizedPrice;
    self.requests[index].fulfilled += toFulfill;
    self.queue.fulfilled = toFulfill;
  }
}
```

Recommendation

Replace = with += as shown:

```
function _fulfill(PooledWithdrawal storage self, uint256 index) internal
    {
  uint256 toFulfill = fulfillable(self, index, self.queue.realized, self.
     queue.fulfilled);
  if (toFulfill > 0) {
    self.requests[index].claimableETH += toFulfill * self.queue.
       realizedPrice;
    self.requests[index].fulfilled += toFulfill;
-   self.queue.fulfilled = toFulfill;
+   self.queue.fulfilled += toFulfill;
  }
}
```

Team Response

Acknowledged, will be mitigated.

[H-01] `processValidators()` Function Can Be Prevented Funds Get Stuck

Severity

High Risk

Description

When there is not enough **`gETH enqueued`** to process validators, it causes a revert and it is not possible to processValidators. Thus, there can be stuck funds even though there are exited validators. Additionally this prevents operators and pool owners from getting their fee.

This is a serious problem. We need to process the amount already requested. But it creates a problem to track the rest and process when more **gETH** comes.

This finding has been identified in close collaboration with the Geode core developer team.

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol#L671

```
function _realizeProcessedEther(
  PooledWithdrawal storage self,
  uint256 processedBalance
) internal {
  .
  .
  self.gETH.burn(address(this), self.POOL_ID, processedgETH);
}

function processValidators(
  PooledWithdrawal storage self,
  bytes[] calldata pubkeys,
  uint256[] calldata beaconBalances,
  uint256[] calldata withdrawnBalances,
  bytes32[][] calldata balanceProofs
) external {
  .
  .
  if (processed > 0) {
    _realizeProcessedEther(self, processed);
  }
}
```

Recommendation

Move **burn()** gETH function from **_realizeProcessedEther()** to **_fulfill()** and **_fulfillBatch()** functions.

```
function _realizeProcessedEther(
  PooledWithdrawal storage self,
  uint256 processedBalance
) internal {
  .
  .
- self.gETH.burn(address(this), self.POOL_ID, processedgETH);
}

function _fulfill(PooledWithdrawal storage self, uint256 index) internal
   {
  .
  .
+ self.gETH.burn(address(this), self.POOL_ID, toFulfill);
}

function _fulfillBatch(
  PooledWithdrawal storage self,
  uint256[] calldata indexes,
  uint256 qRealized,
  uint256 qFulfilled,
  uint256 qPrice
) internal {
  .
  .
+ self.gETH.burn(address(this), self.POOL_ID, qFulfilled - oldFulfilled);
}
```

Team Response

Acknowledged, will be mitigated. We will burn `gETH` on `fulfill` operations instead of `processValidators`.

[H-02] User's Staked ETH Can Be Stuck in the Protocol Through Two Cases

Severity

High Risk

Description

The `deposit()` function in the `Portal.sol` contract allows users to stake their `ETH` into any staking pool that is created. If the user wants to unstake his `ETH` he must make a withdrawal request to the `WithdrawalContract.sol` contract of the given staking pool by calling the `enqueue()` function.

Note: When a staking pool is created, a separate contract is deployed for that staking pool called `WithdrawalContract.sol`.

The impact is primarily financial, as the current implementation for `enqueue()` function in `WithdrawalContract.sol` compels users to initiate a withdrawal with a minimum of `0.05 ETH`.

In contrast `Portal.sol`, the `deposit()` function lacks any restrictions on the amount of `ETH` that can be added to a staking pool.

The following two scenarios show two `Proof of Concepts` of how user's funds can be stuck in the protocol.

Proof of Concept

Scenario 1

```
it("Alice's funds locked - first case", async function () {
// 1. CreatePool - The poolOwner creates a public pool
// 2. CreateOperator - The operatorOwner initiates the operator

// 3. Alice wants to stake 0.04 ETH in this public poll and
// she calls deposit() function, accordingly she will receive 0.04 gETH.
  const aliceFirstDepositAmount = new BN(String(1e18)).muln(0.04); //
     0.04 ETH
  await this.Portal.deposit(publicPoolId, 0, [], 0, MAX_UINT256,
     aliceStaker, {
    from: aliceStaker,
    value: aliceFirstDepositAmount,
  });

// 4. PoolOwner sees Alice's successful transaction and decides to change
    the visibility of the pool to private.
// this means that if a pool is not public, only the controller and the
   whitelisted addresses can deposit/stake.
  await this.Portal.setPoolVisibility(publicPoolId, true, {
    from: poolOwner,
  });


// 5. Expect true to be executed as the pool is private.
expect(await this.Portal.isPrivatePool(publicPoolId)).to.be.equal(true);

// 6. Alice decides to unstake her 0.04 ETH from the "public pool" and
// she calls enqueue() function to queue a withdrawal request into the
   queue.
// Expect revert to be executed as the minimum size of the withdrawal
   request is 0.05 gETH.
await expectRevert(
  this.WML.$enqueue(aliceFirstDepositAmount, "0x", aliceStaker, {
    from: aliceStaker,
  }),
// Actually it's 0.05 ETH (this issue is reported as Informational as
   well).
  "WML:min 0.01 gETH"
);
```

```
// 7. Alice decides to deposit/stake another 0.01 ETH to make a total of
   0.05 ETH because
// this is the minimum size for a withdrawal request.
// Expect revert to be executed because the poolOwner has changed
   visibility to private and
// only the controller and the whitelisted addresses can deposit/stake
   into the private pool.
// 8. Alice's staked funds of 0.04 ETH are locked in the private pool.
const aliceSecondDepositAmount = new BN(String(1e18)).muln(0.01); // 0.01
    ETH
await expectRevert(
  this.Portal.deposit(publicPoolId, 0, [], 0, MAX_UINT256, aliceStaker, {
    from: aliceStaker,
    value: aliceSecondDepositAmount,
  }),
  "SML:no whitelist"
);
});
```

Scenario 2

```
it("Alice's funds locked - second case", async function () {
// 1. CreatePool - The poolOwner creates a public pool
// 2. CreateOperator - The operatorOwner initiates the operator

// 3. Alice wants to stake 0.04 ETH in this public poll and
// she calls deposit() function, accordingly she will receive 0.04 gETH.
  const aliceFirstDepositAmount = new BN(String(1e18)).muln(0.04); //
     0.04 ETH
  await this.Portal.deposit(publicPoolId, 0, [], 0, MAX_UINT256,
     aliceStaker, {
    from: aliceStaker,
    value: aliceFirstDepositAmount,
  });

// 4. The Government decided to pause the Portal contact and accordingly,
// the deposit() function is also paused because there is a whenNotPaused
    modifier
  await this.Portal.pause();
```

```
// 5. Alice decides to unstake her 0.04 ETH from the public pool and
// she calls enqueue() function to queue a withdrawal request into a
   queue.
// Expect revert to be executed as the minimum size of the withdrawal
   request is 0.05 gETH.
  await expectRevert(
    this.WML.$enqueue(aliceFirstDepositAmount, "0x", aliceStaker, {
      from: aliceStaker,
    }),
// Actually it's 0.05 ETH (this issue is reported as Informational as
   well).
    "WML:min 0.01 gETH"
  );

// 6. Alice decides to deposit/stake another 0.01 ETH to make a total of
   0.05 ETH because
// this is the minimum size for a withdrawal request.
// Expect revert to be executed because the deposit() function is paused.
// 7. Alice's staked funds of 0.04 ETH are locked in the public pool.
  const aliceSecondDepositAmount = new BN(String(1e18)).muln(0.01); //
     0.01 ETH
  await expectRevert(
    this.Portal.deposit(publicPoolId, 0, [], 0, MAX_UINT256, aliceStaker,
        {
      from: aliceStaker,
      value: aliceSecondDepositAmount,
    }),
    "Pausable: paused"
  );
});
```

Location of Affected Code

File: contracts/Portal/modules/StakeModule#L404-L425

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

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol#L339

```
function _enqueue(
  PooledWithdrawal storage self,
  uint256 trigger,
  uint256 size,
  address owner
) internal {
  require(size >= MIN_REQUEST_SIZE, "WML:min 0.01 gETH");
  .
  .
}

function enqueue(
  PooledWithdrawal storage self,
  uint256 size,
  bytes calldata pubkey,
  address owner
  ) external {
  uint256 requestedgETH = self.queue.requested;
  _enqueue(self, requestedgETH, size, owner);
  .
  .
}
```

Recommendation

To address this vulnerability, it is crucial to implement a check in the **`deposit()`** function for minimum deposit limit of **`0.05 ETH`** or remove the check for the minimum size of the withdrawal request for **`enqueue()`** function.

Solution one:

File: contracts/Portal/modules/StakeModule/libs/StakeModuleLib.sol#L1023

```
+ uint256 private constant _MIN_DEPOSIT_SIZE = 0.05 ether;

function deposit(
  PooledStaking storage self,
  DSML.IsolatedStorage storage DATASTORE,
  uint256 poolId,
  uint256 mingETH,
  uint256 deadline,
  address receiver
) external returns (uint256 boughtgETH, uint256 mintedgETH) {
  _authenticate(DATASTORE, poolId, false, false, [false, true]);
- require(msg.value > 0, "SML:msg.value cannot be zero");
+ require(msg.value >= _MIN_DEPOSIT_SIZE, "SML:min 0.05 ETH");
  .
  .
}
```

Solution two:

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol#L339

```
function _enqueue(
  PooledWithdrawal storage self,
  uint256 trigger,
  uint256 size,
  address owner
) internal {
- require(size >= MIN_REQUEST_SIZE, "WML:min 0.01 gETH");
  .
  .
}
```

Team Response

Acknowledged, will not be mitigated.

[L-01] `Re-issued` Votes Are Vulnerable

Severity

Low Risk

Description

When the votes of a validator's poll exceed its **`threshold`**, we move the remaining votes to **`commonPoll`**. However, this causes a unique issue:

Since the remaining votes are calculated from **`threshold`**, the same vote that is actually used to bring the validator down is effective on other validators. Thus more validators than needed are called for exit. The remaining funds should be calculated from the **`beaconBalance`** of the validator **`and not the threshold`**.

This finding has been identified in close collaboration with the Geode core developer team.

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol#L241-L261

```
function _checkAndRequestExit(
  PooledWithdrawal storage self,
  bytes calldata pubkey,
  uint256 commonPoll
) internal returns (uint256) {
  uint256 threshold = getValidatorThreshold(self, pubkey);
  uint256 validatorPoll = self.validators[pubkey].poll;
```

```
  if (commonPoll + validatorPoll > threshold) {
    // meaning it can request withdrawal
    if (validatorPoll > threshold) {
      commonPoll += validatorPoll - threshold;
    } else if (threshold > validatorPoll) {
      commonPoll -= threshold - validatorPoll;
    }

    _requestExit(self, pubkey);
  }

  return commonPoll;
}
```

Recommendation

Change the `_checkAndRequestExit()` and `getValidatorThreshold` functions as follows:

```
- function _checkAndRequestExit(
+ function checkAndRequestExit(
  PooledWithdrawal storage self,
  bytes calldata pubkey,
  uint256 commonPoll
- ) internal returns (uint256) {
+ ) public returns (uint256) {
- uint256 threshold = getValidatorThreshold(self, pubkey);
+ (uint256 threshold, uint256 beaconBalancePriced) =
   getValidatorThreshold(self, pubkey);
  uint256 validatorPoll = self.validators[pubkey].poll;

  if (commonPoll + validatorPoll > threshold) {
    // meaning it can request withdrawal
-   if (validatorPoll > threshold) {
+   if (threshold > validatorPoll) {
-     commonPoll += validatorPoll - threshold;
+     commonPoll -= threshold - validatorPoll;
-   } else if (threshold > validatorPoll) {
+   } else if (validatorPoll > beaconBalancePriced) {
-     commonPoll -= threshold - validatorPoll;
+     commonPoll += validatorPoll - beaconBalancePriced;
    }

    _requestExit(self, pubkey);
  }

  return commonPoll;
}
```

```
function getValidatorThreshold(
  PooledWithdrawal storage self,
  bytes calldata pubkey
- ) public view returns (uint256 threshold) {
- uint256 threshold_ETH = (self.validators[pubkey].beaconBalance * self.
   EXIT_THRESHOLD) /
-   PERCENTAGE_DENOMINATOR;
+ ) public view returns (uint256 threshold, uint256 beaconBalancePriced)
   {

  uint256 price = self.gETH.pricePerShare(self.POOL_ID);

- threshold = (((threshold_ETH * self.gETH.denominator()) / price));

+ beaconBalancePriced = ((self.validators[pubkey].beaconBalance *
   gETH_DENOMINATOR));
+ threshold = (beaconBalancePriced * self.EXIT_THRESHOLD) /
   PERCENTAGE_DENOMINATOR / price;
+ beaconBalancePriced = beaconBalancePriced / price;
}
```

Team Response

Acknowledged, will be mitigated.

[L-02] Withdrawal Request Could be Assigned a Wrong New Owner

Severity

Low Risk

Description

The `transferRequest()` function is used to assign a new owner to a request who can later dequeue it. However, since dequeuing ultimately facilitates the claim of ETH, setting a wrong new request owner by mistake can result in a loss of funds.

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol#L424

```
function transferRequest(
  PooledWithdrawal storage self,
  uint256 index,
  address newOwner
) external {
  address oldOwner = self.requests[index].owner;
  require(msg.sender == oldOwner, "WML:not owner");
  require(newOwner != address(0), "WML:cannot transfer to zero address");

  self.requests[index].owner = newOwner;

  emit RequestTransfer(index, oldOwner, newOwner);
}
```

Recommendation

Implement a two-step transfer of the request's ownership. This would include adding a new `address proposedOwner` variable and `proposeNewOwner()` function. The implementation would also require a change in the functionality of the `transferRequest()`.

```
struct Request {
  address owner;
  uint256 trigger;
  uint256 size;
  uint256 fulfilled;
  uint256 claimableETH;
+ address proposedNewOwner;
}

+ function proposeNewOwner(PoolWithdrawal storage self, uint256 index,
    address proposedNewOwner) external {
+   require(msg.sender == self.requests[index].owner, "WML:not owner");
+   require(proposedNewOwner != address(0), "WML:zero address");

+   self.requests[index].proposedNewOwner = proposedNewOwner;
+ }

+ function transferRequest(PoolWithdrawal storage self, uint256 index)
    external {
+   require (msg.sender == self.requests[index].proposedNewOwner, "WML:
    only proposed owner");

+   address oldOwner = self.requests[index].owner;

+   self.requests[index].owner = msg.sender;
+   self.requests[index].proposedNewOwner = address(0);

+   emit RequestTransfer(index, oldOwner, msg.sender);
+ }
```

Team Response

Acknowledged, will not be mitigated.

[I-01] Misleading `require` Message

Severity

Informational

Description

The validation check ensures that `size` is greater or equal to `MIN_REQUEST_SIZE`, and reverts with an incorrect message.

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol#L339

```
require(size >= MIN_REQUEST_SIZE, "WML:min 0.01 gETH");
```

Recommendation

The message should be `WML:min 0.05 gETH`.

Team Response

Mitigated.

[I-02] Missing Event Emission in `processValidators()`

Severity

Informational

Description

Emitting events upon the execution of the core contract is a good practice and can aid the debugging process if the function is used in a multi-call further into the project's development.

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol#L688

```
function processValidators(
  PooledWithdrawal storage self,
  bytes[] calldata pubkeys,
  uint256[] calldata beaconBalances,
  uint256[] calldata withdrawnBalances,
  bytes32[][] calldata balanceProofs
) external {
```

Recommendation

Consider implementing an event, corresponding to the `processValidators` function.

Team Response

Acknowledged, will be implemented.

[I-03] Missing Check for the New Variable in `setExitThreshold`

Severity

Informational

Description

The `setExitThreshold` function lacks a check if the new threshold is the same as the current one. Although this is not harmful, it can lead to unnecessary transactions and gas costs, impacting the user experience.

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol#L271

```
function setExitThreshold(PooledWithdrawal storage self, uint256
    newThreshold) external {
```

Recommendation

Consider adding a check if `self.EXIT_THRESHOLD != newThreshold`;

```
function setExitThreshold(PooledWithdrawal storage self, uint256
    newThreshold) external {
  require(newThreshold >= MIN_EXIT_THRESHOLD, "WML:min threshold is 60%")
     ;
  require(newThreshold <= PERCENTAGE_DENOMINATOR, "WML:max threshold is
     100%");
+ require(self.EXIT_THRESHOLD != newThreshold), "New threshold is the
    same as the current one."

  self.EXIT_THRESHOLD = newThreshold;
  emit NewExitThreshold(newThreshold);
}
```

Team Response

Acknowledged, will not be mitigated.

[I-04] Remove Unused Variables

Severity

Informational

Description

In `WithdrawalModule.sol` variable `size` in `enqueueBatch()` function is unused. In `WithdrawalContract.sol` variable `data` in `initialize()` function is unused.

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/WithdrawalModule.sol#L203

File: contracts/Portal/packages/WithdrawalContract.sol#L64

Recommendation

Consider removing them

Team Response

Mitigated.

[I-05] Use Scientific Notation Rather Than Exponentiation

Severity

Informational

Description

- `1e10` Instead of `10 ** 10`
- `1e6` Instead of `10 ** 6`
- `0.05 ether` Instead of `10 ** 16`

Location of Affected Code

File: contracts/Portal/globals/macros.sol#L5

```
uint256 constant PERCENTAGE_DENOMINATOR = 10 ** 10;
```

File: contracts/Portal/modules/StakeModule/libs/StakeModuleLib.sol#L195

```
uint256 internal constant MAX_ALLOWANCE = 10 ** 6 + 1;
```

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol#L177

```
uint256 constant MIN_REQUEST_SIZE = 5 * 10 ** 16;
```

Recommendation

Consider using scientific notation.

Team Response

Mitigated appropriately.

[I-06] Variables Naming does not Follow the Solidity Style Guide

Severity

Informational

Description

One of the guidelines mentioned in the Style Guide is to name functions and variables in a specific way to improve readability and avoid confusion. By following this, you can achieve consistency in your contract code.

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/WithdrawalModule.sol

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol

Recommendation

Consider using mixedCase for the following variables:

- `Qrealized, Qfulfilled` in the `fulfillable()` function in `WithdrawalModule.sol`.
- `Qrealized`, `Qfulfilled`, `Qprice` in the `_fulfillBatch()` function in `WithdrawalModuleLib.sol`
- `Qrealized`, `Qfulfilled`, `Rtrigger`, `Rsize`, `Rfulfilled`, `Rfloor`, `Rceil` in the `fulfillable()` function in `WithdrawalModuleLib.sol`

Team Response

Mitigated

[I-07] 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/WithdrawalModule/libs/WithdrawalModuleLib.sol#L384

```
//* @dev TODO:  create and use _requestExitBatch to save gas
```

File: contracts/Portal/packages/WithdrawalContract.sol#L20

```
//* TODO: this can be renamed to withdrawalQueue or ValidatorCustodian
```

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

Acknowledged, will be mitigated before the mainnet.

[I-08] Typos

Severity

Informational

Recommendation

```
//WithdawalModuleLibMock -> WithdrawalModuleLibMock

//lenghts -> lengths

//well maintained -> well-maintained

//preventa -> prevent a

//there is remaining votes -> there are remaining votes

//preferance -> preference

//upto -> up to

//Derisked Requests that is -> Derisked Requests that are

//we have came up -> we have come up

//profitablity -> profitability

//distrupting -> disrupting

//became is processed in relative to -> became processed in relation
   to
```

Team Response

Acknowledged, will be mitigated before the mainnet.

[G-01] 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

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol

```
399: for (uint256 i = 0; i < len; i++) {
507: for (uint256 i = 0; i < indexes.length; i++) {
603: for (uint256 i = 0; i < indexes.length; i++) {
704: for (uint256 i = 0; i < pubkeys.length; i++) {
```

Recommendation

Do not initialize variables with their default values.

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

Team Response

Mitigated.

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

Severity

Gas Optimization

Description

When a function parameter of a reference type is marked as read-only, utilizing calldata instead of memory incurs lower gas costs. Calldata serves as an immutable and temporary storage location for function arguments, functioning in a manner largely similar to memory.

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol#L222

```
function _requestExit(PooledWithdrawal storage self, bytes memory pubkey)
    internal {
```

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol#L231

```
function _finalizeExit(PooledWithdrawal storage self, bytes memory pubkey
    ) internal {
```

Recommendation

To optimize gas costs, it is recommended to mark the data type as calldata instead of memory.

Team Response

Acknowledged, will be implemented.

[G-03] Cache Array Length Outside of Loops

Severity

Gas Optimization

Description

In the absence of caching, the Solidity compiler will consistently retrieve the array's length in every iteration. Specifically, for storage arrays, this entails an additional "sload" operation (resulting in 100 extra gas for each iteration, excluding the first one), while for memory arrays, it leads to an additional "mload" operation (resulting in 3 extra gas for each iteration, excluding the first one).

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol

```
507: for (uint256 i = 0; i < indexes.length; i++) {
603: for (uint256 i = 0; i < indexes.length; i++) {
704: for (uint256 i = 0; i < pubkeys.length; i++) {
717: for (uint256 j = 0; j < pubkeys.length; j++) {
```

Recommendation

To optimize gas costs, it is recommended to instantiate a variable in every function that has a for loop, before the loop itself.

Team Response

Mitigated.

[G-04] Revert Strings Can be Shortened

Severity

Gas Optimization

Description

Since the initial proposal from the first audit to use custom errors instead of `require` statements is rather cumbersome for implementation, the current revert strings could be shortened. Keeping them `<= 32 bytes` in length will save gas.

Location of Affected Code

File: contracts/Portal/middlewares/ERC20RebaseMiddleware.sol

```
306: require(currentAllowance >= subtractedValue, "ERC20R: decreased
   allowance below zero");
331: require(from != address(0), "ERC20R: transfer from the zero address"
   );
332: require(to != address(0), "ERC20R: transfer to the zero address");
337: require(fromBalance >= amount, "ERC20R: transfer amount exceeds
   balance");
360: require(owner != address(0), "ERC20R: approve from the zero address"
   );
361: require(spender != address(0), "ERC20R: approve to the zero address"
   );
```

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol

```
340: require(owner != address(0), "WML:owner can not be zero address");
431: require(newOwner != address(0), "WML:cannot transfer to zero address
   ");
572: require(receiver != address(0), "WML:receiver can not be zero
   address");
592: require(receiver != address(0), "WML:receiver can not be zero
   address");
```

Recommendation

Keep revert strings as short as possible. Alternatively, to unify all of them at a certain length, you can represent the revert string as Error codes, for which you can create a dedicate **`Error codes`** section in the documentation, which would include a table with the meaning of all error codes. Inspiration can be taken from [here:] (https://docs.stackup.sh/docs/erc-4337-bundler-rpc-methods#json-rpc-errors).

Team Response

Acknowledged, will be implemented.

[G-05] `++i` Costs Less Gas Than `i++`

Severity

Gas Optimization

Description

The pre-increment operator **`++i`** is a more gas-efficient choice compared to post-increment **`i++`** when employed within for-loops. Utilizing **`++i`** in loops results in a gas savings of 5 units per iteration.

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol

```
399: for (uint256 i = 0; i < len; i++) {
507: for (uint256 i = 0; i < indexes.length; i++) {
603: for (uint256 i = 0; i < indexes.length; i++) {
704: for (uint256 i = 0; i < pubkeys.length; i++) {
717: for (uint256 j = 0; j < pubkeys.length; j++) {
```

Recommendation

Change the post-increment operators in the for-loops to pre-increment ones.

Team Response

We decided to use `unchecked` at the end of the day.

[G-06] Use `!= 0` Instead of `> 0` for Unsigned Integer Comparison

Severity

Gas Optimization

Description

In the context of unsigned integer types, opting for the `!= 0` comparison over `> 0` is typically more gas-efficient. This preference arises from the compiler's capacity to optimize the `!= 0` comparison into a straightforward bitwise operation. In contrast, the `> 0` comparison necessitates an extra subtraction operation. Consequently, choosing `!= 0` can enhance gas efficiency and contribute to lowering the overall contract expenditure.

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol

```
485: if (toFulfill > 0) {
509: if (toFulfill > 0) {
555: require(claimableETH > 0, "WML:not claimable");
661: if (internalPrice > 0) {
663: if (claimable > 0) {
740: if (processed > 0) {
```

Recommendation

Consider changing the `>` to `!=` in the places, outlined above.

Team Response

Acknowledged, will be reconsidered. However, we are planning to bump solidity above `0.8.13`, which mitigates the issue.

[G-07] Do not Calculate Constants

Severity

Gas Optimization

Description

Due to how constant variables are implemented (replacements at compile-time), an expression assigned to a constant variable is recomputed each time the variable is used, which wastes some gas.

Location of Affected Code

File: contracts/Portal/modules/WithdrawalModule/libs/WithdrawalModuleLib.sol#L175

```
uint256 constant MIN_EXIT_THRESHOLD = (6 * PERCENTAGE_DENOMINATOR) / 10;
```

Team Response

Acknowledged, will be reconsidered.

our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Thank you!