Olympus V2 Protocol Audit

Enclosed is the V2 Audit from Omniscia. A URL for an HTML version is also listed below.

Source Code:

https://github.com/OlympusDAO/olympus-contracts/tree/182cfdb29f781c418819fc699c71760ec1dca7e3

URL for Audit Content:

https://omniscia.io/olympus-dao-protocol-v2/

Important Notes:

All findings in this Audit have been alleviated either via code changes or commentary within this audit. There are two Major findings that are listed as not addressed; however, in both cases the Olympus Engineering team deployed alleviations in the final, live contracts.

Protocol V Security Audit

We were tasked with performing a second round audit on the version 2 implementation of the Olympus DAO protocol composed of a complex system architecture involving a triple token system, an LP-based bond system, and utility contracts for incentivizing the use of all three token types.

Over the course of the audit, we were able to pinpoint potentially harmful arbitrage opportunities that can arise in the conversion between the three tokens as well as a potential under-pricing flaw in the bond creation mechanism that if exploited could cause a bond to be priced at a very low value and thus cause a significant evaluation of an otherwise small deposit.

In addition to logical flaws, we identified several optimizations that can be applied to the codebase that we urge the Olympus DAO team to consider. Overall, the codebase appears to be at an unpolished state and can be significantly improved in terms of styling, consistency, and documentation. For the former, we advise a linting plugin to be enforced on the codebase to greatly increase its readability.

Another important point that should be raised about the codebase is the over-reliance on good faith of the various authorized operators in the protocol. As an example, the terms of a bond are not validated and permit arbitrary values for all terms whilst they are only set by the guardian of the protocol. As we have expressed in some of the exhibits, we advise the Olympus DAO team to attempt to further decentralize the operation of the protocol by introducing new sanitization checks restricting the authorative actions of the privileged roles of the system.

Files in Scope	Repository	Commit(s)
Address.sol (ADD)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
BondTeller.sol (BTR)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f

Files in Scope	Repository	Commit(s)
BondDepository.sol (BDY)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
Counters.sol (COU)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
ERC20.sol (ERC)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
ERC20Permit.sol (ERP)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
EnumerableSet.sol (EST)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
FullMath.sol (FMH)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
FixedPoint.sol (FPT)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
Guardable.sol (GUA)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
Governable.sol (GOV)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
GovernorAlpha.sol (GAA)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
GovernorOHMegaDelegate.sol (GOH)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f

Files in Scope	Repository	Commit(s)
GovernorOHMegaDelegator.sol (GOM)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
GovernorOHMegaInterfaces.sol (GOI)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
ManagerOwnable.sol (MOE)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
Ownable.sol (OWN)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
OlympusERC20.sol (OER)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
OlympusTokenMigrator.sol (OTM)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
Staking.sol (STA)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
SafeMath.sol (SMH)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
SafeERC20.sol (SER)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
StakingDistributor.sol (SDR)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
StandardBondingCalculator.sol (SBC)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f

Files in Scope	Repository	Commit(s)
Timelock.sol (TIM)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
Treasury.sol (TRE)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
VaultOwned.sol (VOD)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
gOHM.sol (OHM)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f
sOlympusERC20.sol (OEC)	olympus-contracts	61f3d44487, 21fe403ed7, 182cfdb29f

During the audit, we filtered and validated a total of **10 findings utilizing static analysis** tools as well as identified a total of **73 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they introduce potential misbehaviours of the system as well as exploits.

The list below covers each segment of the audit in depth and links to the respective chapter of the report:

- 💻 **Compilation**
- 🔍 **Static Analysis**
- 👁️ **Manual Review**
- 🖌️ **Code Style**

NEXT
Compilation ›

Compilation

The project utilizes `hardhat` as its development pipeline tool, containing an array of tests and scripts coded in TypeScript.

To compile the project, the `compile` command needs to be issued via the `npx` CLI tool to `hardhat`:

```bash
npx hardhat compile
```

The `hardhat` tool automatically selects between Solidity versions `0.8.9`, `0.7.5`, and `0.5.16` based on the version specified within the `hardhat.config.ts` file as well as the `pragma` statement of the contract being currently compiled.

The project contains discrepancies with regards to the Solidity version used, however, they are located in external dependencies of the project and as such can be safely ignored.

The Olympus DAO team has locked the `pragma` statements to `0.7.5`, the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `hardhat` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

PREV
Introduction

NEXT
Static Analysis

Static Analysis

The execution of our static analysis toolkit identified **420 potential issues** within the codebase of which **407 were ruled out to be false positives** or negligible findings.

The remaining **13 issues** were validated and grouped and formalized into the **10 exhibits** that follow:

ID	Severity	Addressed	Title
BTR-01S	Major	Yes	Inexistent Initialization of Member
BTR-02S	Informational	Yes	Improper Inheritence
SDR-01S	Informational	Yes	Improper Inheritence
SDR-02S	Informational	Yes	Undocumented Value Literal
TRE-01S	Minor	Yes	Improper Usage of EIP-20 Transfer
TRE-02S	Informational	Yes	Literal Equality of `bool` Variables
VOD-01S	Minor	Yes	Inexistent Validation of Address Argument
VOD-02S	Informational	Yes	Inexistent Emission of Event
OEC-01S	Informational	Yes	Illegible Numeric Literal
OEC-02S	Informational	Yes	Improper Inheritence

BondTeller Static Analysis Findings

ON THIS PAGE

BTR-01S: Inexistent Initialization of Member

Type	Severity	Location
Logical Fault	Major ●	BondTeller.sol:L57

Description:

The `policy` member of the contract is never initialized.

Example:

contracts/BondTeller.sol

SOL

Copy

```sol
57  address public policy;
58
59  /* ========== CONSTRUCTOR ========== */
60
61  constructor(
62      address _depository,
63      address _staking,
64      address _treasury,
65      address _OHM,
66      address _sOHM,
67      address _gOHM
68  ) {
69      require(_depository != address(0));
```

```
70      depository = _depository;
71      require(_staking != address(0));
72      staking = IStaking(_staking);
73      require(_treasury != address(0));
74      treasury = ITreasury(_treasury);
75      require(_OHM != address(0));
76      OHM = IERC20(_OHM);
77      require(_sOHM != address(0));
78      sOHM = IERC20(_sOHM);
79      require(_gOHM != address(0));
80      gOHM = IgOHM(_gOHM);
81      IERC20(_OHM).approve(_staking, 1e27); // saves gas
82  }
```

Recommendation:

We advise it to be initialized to ensure the `setFEReward` function can be invoked.

Alleviation:

The `policy` member has now been removed from the contract, thereby nullifying this exhibit.

View Fix on GitHub

BTR-02S: Improper Inheritence

Type	Severity	Location
Code Style	Informational	BondTeller.sol:L12

Description:

The `BondTeller` contract complies with the `ITeller` interface of the codebase yet does not inherit it.

Example:

contracts/BondTeller.sol

SOL Copy

```
12  contract BondTeller {
```

Recommendation:

We advise the contract to properly inherit it ensuring consistency and maintainability across the codebase.

Alleviation:

The contract now properly inherits the `ITeller` interface.

View Fix on GitHub

PREV
Code Style

NEXT
StakingDistributor.sol (SDR-S)

StakingDistributor Static Analysis Findings

ON THIS PAGE

SDR-01S: Improper Inheritence

Type	Severity	Location
Code Style	Informational ●	StakingDistributor.sol:L12

Description:

The `Distributor` contract complies with the `IDistributor` interface of the codebase yet does not inherit it.

Example:

contracts/StakingDistributor.sol

SOL Copy

```sol
13  contract Distributor is Governable, Guardable {
```

Recommendation:

We advise the contract to properly inherit it ensuring consistency and maintainability across the codebase.

Alleviation:

The contract now properly inherits the `IDistributor` interface.

View Fix on GitHub

SDR-02S: Undocumented Value Literal

Type	Severity	Location
Code Style	Informational ●	StakingDistributor.sol:L107

Description:

The value literal `1000000` is meant to be used as the rate divisor for a particular reward distribution, however, it is undocumented and unclearly depicted.

Example:

contracts/StakingDistributor.sol

SOL Copy

```sol
107 return OHM.totalSupply().mul(_rate).div(1000000);
```

Recommendation:

We advise the special underscore (_) separator to be applied to it (i.e. `1000000` would become `1_000_000`) and we advise the value to be set to a contract-level `constant` as it will be useful for other exhibits and general logic checks of the codebase.

Alleviation:

The numeric literal was relocated to a contract-level `immutable` declaration thereby alleviating this exhibit.

View Fix on GitHub

PREV
BondTeller.sol (BTR-S)

NEXT
Treasury.sol (TRE-S)

Treasury Static Analysis Findings

ON THIS PAGE

TRE-01S: Improper Usage of EIP-20 Transfer

Type	Severity	Location
Standard Conformity	Minor •	Treasury.sol:L160

Description:

The **EIP-20** standard denotes that callers MUST NOT assume that `false` is never returned in `transfer` invocations and should be able to gracefully handle the returned `bool` of the function invocation.

Example:

contracts/Treasury.sol

SOL

Copy

```
160 IERC20(_token).transfer(msg.sender, _amount);
```

Recommendation:

As certain tokens are not compliant with the standard, we advise the usage of a wrapper library such as `SafeERC20` of OpenZeppelin that opportunistically evaluates the yielded `bool` if it exists.

Alleviation:

The linked EIP-20 `transfer` call is now properly wrapped in its `safe`-prefixed equivalent by OpenZeppelin's `SafeERC20` library.

View Fix on GitHub

TRE-02S: Literal Equality of `bool` Variables

Type	Severity	Location
Code Style	Informational ●	Treasury.sol:L128

Description:

The linked statement performs a direct comparison between a `bool` variable and a `bool` literal.

Example:

contracts/Treasury.sol

SOL Copy

```
128 require(permissions[STATUS.RESERVESPENDER][msg.sender] == true, "Not approved
```

Recommendation:

We advise the `bool` variable to be utilized directly either in its normal or negated (`!`) form, depending on the `bool` literal it was being compared to.

Alleviation:

The `bool` variable is now utilized directly in the `require` check.

View Fix on GitHub

PREV
StakingDistributor.sol (SDR-S)

NEXT
VaultOwned.sol (VOD-S)

VaultOwned Static Analysis Findings

ON THIS PAGE

VOD-01S: Inexistent Validation of Address Argument

Type	Severity	Location
Input Sanitization	Minor •	VaultOwned.sol:L10-L14

Description:

The linked function contains an `address` argument that is not properly sanitized against the zero-address.

Example:

contracts/types/VaultOwned.sol

SOL

Copy

```sol
10  function setVault( address vault_ ) external onlyOwner() returns ( bool ) {
11    _vault = vault_;
12
13    return true;
14  }
```

Recommendation:

We advise it to be sanitized so to avoid potential misconfigurations of the contract.

Alleviation:

The Olympus DAO team considered this exhibit but opted not to apply any remediation for it.

View Fix on GitHub

VOD-02S: Inexistent Emission of Event

Type	Severity	Location
Input Sanitization	Informational	VaultOwned.sol:L10-L14

Description:

The linked function adjusts a sensitive contract variable without emitting a corresponding `event`.

Example:

contracts/types/VaultOwned.sol

SOL

Copy

```sol
10  function setVault( address vault_ ) external onlyOwner() returns ( bool ) {
11    _vault = vault_;
12
13    return true;
14  }
```

Recommendation:

We advise an `event` to be coded for the action and emitted whenever it is executed to ensure off-chain observers of the contracts can properly sync their data points.

Alleviation:

The Olympus DAO team considered this exhibit but opted not to apply any remediation for it.

View Fix on GitHub

PREV
Treasury.sol (TRE-S)

NEXT

sOlympusERC20 Static Analysis Findings

ON THIS PAGE

OEC-01S: Illegible Numeric Literal

Type	Severity	Location
Code Style	Informational	sOlympusERC20.sol:L55

Description:

The linked variable contains a numeric literal with too many digits and no separator.

Example:

contracts/sOlympusERC20.sol

SOL

Copy

```sol
55  uint256 private constant INITIAL_FRAGMENTS_SUPPLY = 5000000 * 10**9;
```

Recommendation:

We advise the special numeric separator (_) to be used to discern per thousand units (i.e. `10000` becomes `10_000`), increasing the legibility of the codebase.

Alleviation:

The underscore (_) numeric separator was properly introduced to the linked variable.

View Fix on GitHub

OEC-02S: Improper Inheritence

Type	Severity	Location
Code Style	Informational	sOlympusERC20.sol:L12

Description:

The `sOlympus` contract complies with the `IsOHM` interface of the codebase yet does not inherit it.

Example:

contracts/sOlympusERC20.sol

SOL Copy

```sol
12  contract sOlympus is ERC20Permit {
```

Recommendation:

We advise the contract to properly inherit it ensuring consistency and maintainability across the codebase.

Alleviation:

The now properly inherits the `IsOHM` interface.

View Fix on GitHub

PREV
VaultOwned.sol (VOD-S)

NEXT
BondDepository.sol (BDY-M)

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in the version 2 iteration of the Olympus DAO protocol.

As the project at hand implements a complex system architecture of a three token system and a bond pricing mechanism, intricate care was put into ensuring that the **flow of funds within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed certain misconceptions** within the system which could have had **severe ramifications** to its overall operation when exploited under the right circumstances, however, they were conveyed ahead of time to the Olympus DAO team to be **promptly remediated**.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to a certain extent, however, we strongly recommend the documentation of the project to be expanded at certain complex points such as the mathematical operations surrounding the pricing of debt ratios utilizing the undocumented `decode112with18` function.

A total of **73 findings** were identified over the course of the manual review of which **39 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
BDY-01M	Major	No	Improper Bond Price Assumption
BDY-02M	Medium	No	Inexistent Validation of Terms

BondDepository Manual Review Findings

ON THIS PAGE

BDY-01M: Improper Bond Price Assumption

Type	Severity	Location
Logical Fault	Major ●	BondDepository.sol:L328-L341

Description:

The `_bondPrice` adjusts the on-chain minimum price of a bond to `0` if the price of the bond is exceeded, however, the `debtRatio` variable relies on the total supply of OHM which is fully manipulateable and thus can cause the price of a bond to increase above the minimum temporarily by taking advantage of the `debtRatio` reliance on `OHM.totalSupply`.

Example:

contracts/BondDepository.sol

SOL

Copy

```sol
328 /**
329  * @notice calculate current bond price and remove floor if above
330  * @param _BID uint
331  * @return price_ uint
332  */
333 function _bondPrice(uint256 _BID) internal returns (uint256 price_) {
334   Bond memory info = bonds[_BID];
335   price_ = info.terms.controlVariable.mul(debtRatio(_BID)).add(1000000000).di
```

```
336   if (price_ < info.terms.minimumPrice) {
337     price_ = info.terms.minimumPrice;
338   } else if (info.terms.minimumPrice != 0) {
339     bonds[_BID].terms.minimumPrice = 0;
340   }
341 }
342
343 /**
344  * @notice converts bond price to DAI value
345  * @param _BID uint
346  * @return price_ uint
347  */
348 function bondPriceInUSD(uint256 _BID) public view returns (uint256 price_) {
349   Bond memory bond = bonds[_BID];
350   if (address(bond.calculator) != address(0)) {
351     price_ = bondPrice(_BID).mul(bond.calculator.markdown(address(bond.princi
352   } else {
353     price_ = bondPrice(_BID).mul(10**IERC20Metadata(address(bond.principal)).
354   }
355 }
356
357 // DEBT
358
359 /**
360  * @notice calculate current ratio of debt to OHM supply
361  * @param _BID uint
362  * @return debtRatio_ uint
363  */
364 function debtRatio(uint256 _BID) public view returns (uint256 debtRatio_) {
365   debtRatio_ = FixedPoint.fraction(currentDebt(_BID).mul(1e9), OHM.totalSuppl
366 }
```

Recommendation:

We advise the function to not adjust the minimum price as it can lead to the pricing of a bond becoming less-than-minimum under the right circumstances.

Alleviation:

The Olympus DAO team considered this exhibit but decided to retain the current behaviour of the code in place.

BDY-02M: Inexistent Validation of Terms

Type	Severity	Location
Input Sanitization	Medium	BondDepository.sol:L97-L129

Description:

All arguments of the `setTerms` function are not sanitized and as such can be arbitrarily set to values that may be illogical or result in exploits, such as a `conclusion` that is beyond the `expiration` of the bond and other similar issues.

Example:

contracts/BondDepository.sol

SOL

Copy

```sol
97  /**
98   * @notice set minimum price for new bond
99   * @param _id uint
100  * @param _controlVariable uint
101  * @param _fixedTerm bool
102  * @param _vestingTerm uint
103  * @param _expiration uint
104  * @param _conclusion uint
105  * @param _minimumPrice uint
106  * @param _maxPayout uint
107  * @param _maxDebt uint
108  * @param _initialDebt uint
109  */
110 function setTerms(
111   uint256 _id,
112   uint256 _controlVariable,
113   bool _fixedTerm,
114   uint256 _vestingTerm,
115   uint256 _expiration,
116   uint256 _conclusion,
117   uint256 _minimumPrice,
118   uint256 _maxPayout,
```

```
119     uint256 _maxDebt,
120     uint256 _initialDebt
121 ) external onlyGuardian {
122     require(!bonds[_id].termsSet, "Already set");
123 
124     Terms memory terms = Terms({controlVariable: _controlVariable, fixedTerm: _
125 
126     bonds[_id].terms = terms;
127     bonds[_id].totalDebt = _initialDebt;
128     bonds[_id].termsSet = true;
129 }
```

Recommendation:

We strongly recommend some form of input sanitization to be enforced on the bond terms as in the current state the guardian has significant control over the protocol's normal operation.

Alleviation:

The Olympus DAO team considered this exhibit but decided to retain the current behaviour of the code in place.

BDY-03M: Inexplicable Optional Value of Decay

Type	Severity	Location
Logical Fault	Medium	BondDepository.sol:L390-L402

Description:

The `debtDecay` function relies on the `vestingTerm` variable of the `Term` struct which is meant to be an optional variable based on the documentation of the struct.

Example:

contracts/BondDepository.sol

SOL

Copy

```sol
390 /**
391  * @notice amount to decay total debt by
392  * @param _BID uint
393  * @return decay_ uint
394  */
395 function debtDecay(uint256 _BID) public view returns (uint256 decay_) {
396   Bond memory bond = bonds[_BID];
397   uint256 blocksSinceLast = block.number.sub(bond.lastDecay);
398   decay_ = bond.totalDebt.mul(blocksSinceLast).div(bond.terms.vestingTerm);
399   if (decay_ > bond.totalDebt) {
400     decay_ = bond.totalDebt;
401   }
402 }
```

Recommendation:

We advise its purpose to be better defined in the struct declaration as bonds without a fixed term are possible, causing decay to misbehave.

Alleviation:

The Olympus DAO team considered this exhibit but decided to retain the current behaviour of the code in place

the code in place.

PREV
sOlympusERC20.sol (OEC-S)

NEXT
BondTeller.sol (BTR-M)

BondTeller Manual Review Findings

ON THIS PAGE

BTR-01M: Confusion of Value Denominations

Type	Severity	Location
Logical Fault	Major ●	BondTeller.sol:L232-L245

Description:

The `percentVestedFor` function assumes the `bond.vested` value to be a timestamp yet it represents a block.

Example:

contracts/BondTeller.sol

SOL

Copy

```sol
232 /**
233  * @notice calculate how far into vesting a depositor is
234  * @param _bonder address
235  * @param _index uint
236  * @return percentVested_ uint
237  */
238 function percentVestedFor(address _bonder, uint256 _index) public view return
239     Bond memory bond = bonderInfo[_bonder][_index];
240
```

```
241     uint256 timeSince = block.timestamp.sub(bond.created);
242     uint256 term = bond.vested.sub(bond.created);
243
244     percentVested_ = timeSince.mul(1e9).div(term);
245 }
```

Recommendation:

We strongly recommend the function to be adjusted as it should be inoperable in its current state.

Alleviation:

After discussion with the Olympus DAO team, both units are meant to represent a timestamp and as such this exhibit can be considered null.

View Fix on GitHub

BTR-02M: Artificial Inflation Mechanism

Type	Severity	Location
Logical Fault	Medium	BondTeller.sol:L104, L110

Description:

The `newBond` function is meant to award bond creators with a `feReward` that is added to the original payout of a bond, however, this action introduces a secondary level of inflation that can also be redirected to the user themselves and could result in a significant issue if `feReward` is close to the minimum payout as users are incentivized to break their bonds into multiple smaller ones and set the `_feo` as themselves.

Example:

contracts/BondTeller.sol

SOL

Copy

```sol
86  /**
87   * @notice add new bond payout to user data
88   * @param _bonder address
89   * @param _principal address
90   * @param _principalPaid uint
91   * @param _payout uint
92   * @param _expires uint
93   * @param _feo address
94   * @return index_ uint
95   */
96  function newBond(
97      address _bonder,
98      address _principal,
99      uint256 _principalPaid,
100     uint256 _payout,
101     uint256 _expires,
102     address _feo
103 ) external onlyDepository returns (uint256 index_) {
104     treasury.mint(address(this), _payout.add(feReward));
105
```

```
106     OHM.approve(address(staking), _payout); // approve staking payout
107
108     staking.stake(_payout, address(this), true, true);
109
110     FERs[_feo] = FERs[_feo].add(feReward); // FE operator takes fee
111
112     index_ = bonderInfo[_bonder].length;
113
114     // store bond & stake payout
115     bonderInfo[_bonder].push(Bond({principal: _principal, principalPaid: _pri
116 }
```

Recommendation:

We strongly recommend this aspect of the protocol to be revised. Some potential solutions would be to retain an address of whitelisted front end operators that rewards can be re-directed to and having the `feReward` be based on a percentage rather than a static value.

Alleviation:

The Olympus DAO team considered this exhibit but opted not to apply a remediation for it as they consider the current mechanism sufficiently secure.

BTR-03M: Inexistent Redemption of FEO Fees

Type	Severity	Location
Logical Fault	Minor	BondTeller.sol:L110

Description:

The front-end operator (FEO) fees are stored within the contract and are minted, however, there is no way to claim them by the FEOs.

Example:

contracts/BondTeller.sol

SOL

Copy

```sol
/**
 * @notice add new bond payout to user data
 * @param _bonder address
 * @param _principal address
 * @param _principalPaid uint
 * @param _payout uint
 * @param _expires uint
 * @param _feo address
 * @return index_ uint
 */
function newBond(
    address _bonder,
    address _principal,
    uint256 _principalPaid,
    uint256 _payout,
    uint256 _expires,
    address _feo
) external onlyDepository returns (uint256 index_) {
    treasury.mint(address(this), _payout.add(feReward));

    OHM.approve(address(staking), _payout); // approve staking payout

    staking.stake(_payout, address(this), true, true);

```

```
110     FERs[_feo] = FERs[_feo].add(feReward); // FE operator takes fee
111
112     index_ = bonderInfo[_bonder].length;
113
114     // store bond & stake payout
115     bonderInfo[_bonder].push(Bond({principal: _principal, principalPaid: _pri
116 }
```

Recommendation:

We advise some form of pull-pattern to be applied where FEOs are able to invoke a function to retrieve all fees they have accumulated.

Alleviation:

A fee redemption mechanism was introduced in the codebase in the form of the `getReward` function that transfers the necessary OHM outward to the user.

View Fix on GitHub

BTR-04M: Inexistent Validation of Non-Zero Redemption

Type	Severity	Location
Input Sanitization	Minor ●	BondTeller.sol:L148

Description:

The `redeem` function should validate that a non-zero redemption is being performed.

Example:

contracts/BondTeller.sol

SOL

Copy

```sol
130 /**
131  *  @notice redeem bond for user
132  *  @param _bonder address
133  *  @param _indexes calldata uint[]
134  *  @return uint
135  */
136 function redeem(address _bonder, uint256[] memory _indexes) public returns (u
137     uint256 dues;
138     for (uint256 i = 0; i < _indexes.length; i++) {
139         Bond memory info = bonderInfo[_bonder][_indexes[i]];
140 
141         if (pendingFor(_bonder, _indexes[i]) != 0) {
142             bonderInfo[_bonder][_indexes[i]].redeemed = block.timestamp; // m
143 
144             dues = dues.add(info.payout);
145         }
146     }
147 
148     dues = gOHM.balanceFrom(dues);
149 
150     emit Redeemed(_bonder, dues);
151     pay(_bonder, dues);
152     return dues;
153 }
```

Recommendation:

We advise this to be done so by ensuring that `dues` is non-zero beyond conversion.

Alleviation:

The Olympus DAO team considered this exhibit but decided to retain the current behaviour of the code in place.

PREV
BondDepository.sol (BDY-M)

NEXT
ERC20.sol (ERC-M)

ERC20 Manual Review Findings

ON THIS PAGE

ERC-01M: Non-Standard Mint Implementation

Type	Severity	Location
Standard Conformity	Minor •	ERC20.sol:L95-L101

Description:

The `_mint` function of the `ERC20` implementation invokes the `_beforeTokenTransfer` hook and the `Transfer` event with the `from` argument being the `address(this)`, an adaptation that will cause off-chain explorers to misbehave.

Example:

contracts/types/ERC20.sol

SOL Copy

```sol
function _mint(address account_, uint256 ammount_) internal virtual {
    require(account_ != address(0), "ERC20: mint to the zero address");
    _beforeTokenTransfer(address( this ), account_, ammount_);
    _totalSupply = _totalSupply.add(ammount_);
    _balances[account_] = _balances[account_].add(ammount_);
    emit Transfer(address( this ), account_, ammount_);
}
```

Recommendation:

We advise both instances to be replaced by the canonical `address(0)` to enable proper

interfacing of block explorers.

Alleviation:

The hook and event instances now properly use the zero address instead of the self address to indicate the origin of the "transfer" operation, thereby alleviating this exhibit.

View Fix on GitHub

PREV
BondTeller.sol (BTR-M)

NEXT
ERC20Permit.sol (ERP-M)

ERC20Permit Manual Review Findings

ON THIS PAGE

ERP-01M: Insecure Elliptic Curve Recovery Mechanism

Type	Severity	Location
Language Specific	Medium	ERC20Permit.sol:L55

Description:

The `ecrecover` function is a low-level cryptographic function that should be utilized after appropriate sanitizations have been enforced on its arguments, namely on the `s` and `v` values. This is due to the inherent trait of the curve to be symmetrical on the x-axis and thus permitting signatures to be replayed with the same `x` value (`r`) but a different `y` value (`s`).

Example:

contracts/types/ERC20Permit.sol

SOL

Copy

```sol
/**
 * @dev See {IERC2612Permit-permit}.
 *
 */
function permit(
    address owner,
    address spender,
    uint256 amount,
    uint256 deadline,
```

```
44      uint8 v,
45      bytes32 r,
46      bytes32 s
47  ) public virtual override {
48      require(block.timestamp <= deadline, "Permit: expired deadline");
49
50      bytes32 hashStruct =
51          keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, amount, _nonces
52
53      bytes32 _hash = keccak256(abi.encodePacked(uint16(0x1901), DOMAIN_SEPARAT
54
55      address signer = ecrecover(_hash, v, r, s);
56      require(signer != address(0) && signer == owner, "ZeroSwapPermit: Invalid
57
58      _nonces[owner].increment();
59      _approve(owner, spender, amount);
60  }
```

Recommendation:

We advise them to be sanitized by ensuring that `v` is equal to either `27` or `28` (`v ∈ {27, 28}`) and to ensure that `s` is existent in the lower half order of the elliptic curve (`0 < s < secp256k1n ÷ 2 + 1`) by ensuring it is less than `0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A1`. A reference implementation of those checks can be observed in the **ECDSA** library of OpenZeppelin and the rationale behind those restrictions exists within **Appendix F of the Yellow Paper**.

Alleviation:

The code was adjusted to instead utilize the `ECDSA` library from OpenZeppelin directly that imposes the necessary sanitizations on the `v` and `s` arguments of the `ecrecover` function.

View Fix on GitHub

ERP-02M: Cross-Chain Signature Replay Attack Susceptibility

Type	Severity	Location
Language Specific	Minor •	ERC20Permit.sol:L19-L33

Description:

The `DOMAIN_SEPARATOR` used in conjunction with the **EIP-712** standard is calculated once during the `constructor` of the contract. As this calculation involves the execution context's `chainid`, blockchain forks will allow signatures to be replayed across chains as the `chainid` is consequently not validated.

Example:

contracts/types/ERC20Permit.sol

SOL Copy

```sol
constructor() {

    uint256 chainID;
    assembly {
        chainID := chainid()
    }

    DOMAIN_SEPARATOR = keccak256(abi.encode(
        keccak256("EIP712Domain(string name,string version,uint256 chainId,ad
        keccak256(bytes(name())),
        keccak256(bytes("1")), // Version
        chainID,
        address(this)
    ));
}
```

Recommendation:

We advise a caching mechanism to be imposed here instead whereby the `DOMAIN_SEPARATOR` is stored to an `immutable` contract-level variable and is utilized only when the cached `chainid`

(also stored in an `immutable` variable) matches the current execution context's `chainid`. A reference implementation of this paradigm can be observed at the `draft-EIP712.sol` **implementation of OpenZeppelin**.

Alleviation:

The code took inspiration from the `draft-EIP712.sol` implementation of OpenZeppelin and now uses a caching system that dynamically calculates the separator as needed depending on the evaluation of the current `chainID`, thereby alleviating this exhibit.

View Fix on GitHub

PREV
ERC20.sol (ERC-M)

NEXT
FixedPoint.sol (FPT-M)

FixedPoint Manual Review Findings

ON THIS PAGE

FPT-01M: Potentially Invalid Implementation

Type	Severity	Location
Logical Fault	Major	FixedPoint.sol:L110-L113

Description:

The `decode112with18` function is non-standard, utilizes a value literal and cannot be validated as to its purpose.

Example:

contracts/libraries/FixedPoint.sol

SOL

Copy

```sol
110 function decode112with18(uq112x112 memory self) internal pure returns (uint)
111
112     return uint(self._x) / 5192296858534827;
113 }
```

Recommendation:

We advise it to be extensively documented as in its current state it is ambiguous. This finding will be adjusted accordingly when the code segment has been properly documented.

Alleviation:

The Olympus DAO team has not provided a response for this exhibit yet.

PREV
ERC20Permit.sol (ERP-M)

NEXT
Governable.sol (GOV-M)

Governable Manual Review Findings

ON THIS PAGE

GOV-01M: Improper Governor Renouncation

Type	Severity	Location
Logical Fault	Medium	Governable.sol:L31-L34

Description:

The `renounceGovernor` function deletes the current `_governor` in place, however, the `_newGovernor` remains especially so when the `pullGovernor` function is invoked once as it does not delete the previous entry. This would permit governorship to be re-instated even after it has been renounced on a particular contract.

Example:

contracts/types/Governable.sol

SOL

Copy

```sol
function renounceGovernor() public virtual override onlyGovernor() {
    emit GovernorPushed( _governor, address(0) );
    _governor = address(0);
}
```

Recommendation:

We strongly recommend the `_newGovernor` to also be deleted when `renounceGovernor` is invoked to prevent improper state transitions.

Alleviation:

The new governor is properly deleted when governorship is renounced.

View Fix on GitHub

GOV-02M: Incorrect Event Emitted

Type	Severity	Location
Logical Fault	Minor ●	Governable.sol:L33

Description:

The `renounceGovernor` function emits the `GovernorPushed` event when the one in place is renounced, however, it should `emit` the `GovernorPulled` event instead.

Example:

contracts/types/Governable.sol

SOL Copy

```sol
16  constructor () {
17      _governor = msg.sender;
18      emit GovernorPulled( address(0), _governor );
19  }
20
21  /* ========== GOVERNOR ========== */
22
23  function governor() public view override returns (address) {
24      return _governor;
25  }
26
27  modifier onlyGovernor() {
28      require( _governor == msg.sender, "Governable: caller is not the governor
29      _;
30  }
31
32  function renounceGovernor() public virtual override onlyGovernor() {
33      emit GovernorPushed( _governor, address(0) );
34      _governor = address(0);
35  }
36
37  function pushGovernor( address newGovernor_ ) public virtual override onlyGov
38      require( newGovernor_ != address(0), "Governable: new governor is the zer
39      emit GovernorPushed(  governor, newGovernor  );
```

```
40      _newGovernor = newGovernor_;
41  }
42
43  function pullGovernor() public virtual override {
44      require( msg.sender == _newGovernor, "Governable: must be new governor to
45      emit GovernorPulled( _governor, _newGovernor );
46      _governor = _newGovernor;
47  }
```

Recommendation:

We advise it to do so as the latter `event` is utilized in functions like `pullGovernor` and the `constructor`.

Alleviation:

The `GovernorPulled` event is now properly emitted within the `renounceGovernor` function.

View Fix on GitHub

GOV-03M: Potentially Restrictive Functionality

Type	Severity	Location
Logical Fault	Minor ●	Governable.sol:L38

Description:

The `pushGovernor` function validates that the `newGovernor_` being set is not the zero address, however, in doing so it will prevent a pending governor to be overwritten with a zero entry.

Example:

contracts/types/Governable.sol

SOL

Copy

```
37  function pushGovernor( address newGovernor_ ) public virtual override onlyGov
38      require( newGovernor_ != address(0), "Governable: new governor is the zer
39      emit GovernorPushed( _governor, newGovernor_ );
40      _newGovernor = newGovernor_;
41  }
```

Recommendation:

We advise this check to either be omitted entirely or only be evaluated when `_newGovernor` is itself equal to the zero-address as otherwise, an incorrectly set `_newGovernor` will not be able to be deleted.

Alleviation:

The restrictive `require` check is now omitted from the codebase.

View Fix on GitHub

PREV
FixedPoint.sol (FPT-M)

NEXT
GovernorAlpha.sol (GAA-M)

GovernorAlpha Manual Review Findings

ON THIS PAGE

GAA-01M: Improper Percentage Documented

Type	Severity	Location
Logical Fault	Informational	GovernorAlpha.sol:L14

Description:

The `proposalThresholdPercent` is meant to showcase the percentage of voting power required to make a proposal, however, it is incorrectly documented as 1.00% when in reality it is 10% when the divisor of the `getVotesFromPercentOfsOHMSupply` function is taken into account.

Example:

contracts/governance/GovernorAlpha.sol

SOL

Copy

```sol
/// @notice The maximum setable proposal threshold percent
/// @notice change from original contract
function proposalThresholdPercent() public pure returns (uint) { return 10000
```

Recommendation:

We advise the documentation or value itself to be properly remediated depending on the canonical value of the proposal threshold.

Alleviation:

The value was corrected to the 1.00% indicated by its corresponding comments.

View Fix on GitHub

PREV
Governable.sol (GOV-M)

NEXT
Guardable.sol (GUA-M)

Guardable Manual Review Findings

ON THIS PAGE

GUA-01M: Improper Guardian Renouncation

Type	Severity	Location
Logical Fault	Medium	Guardable.sol:L31-L34

Description:

The `renounceGuardian` function deletes the current `_guardian` in place, however, the `_newGuardian` remains especially so when the `pullGuardian` function is invoked once as it does not delete the previous entry. This would permit guardianship to be re-instated even after it has been renounced on a particular contract.

Example:

contracts/types/Guardable.sol

SOL

Copy

```sol
31  function renounceGuardian() public virtual override onlyGuardian() {
32      emit GuardianPushed( _guardian, address(0) );
33      _guardian = address(0);
34  }
```

Recommendation:

We strongly recommend the `_newGuardian` to also be deleted when `renounceGuardian` is invoked to prevent improper state transitions.

Alleviation:

The new guardian is properly deleted when guardianship is renounced.

View Fix on GitHub

GUA-02M: Incorrect Event Emitted

Type	Severity	Location
Logical Fault	Minor ●	Guardable.sol:L32

Description:

The `renounceGuardian` function emits the `GuardianPushed` event when the one in place is renounced, however, it should `emit` the `GuardianPulled` event instead.

Example:

contracts/types/Guardable.sol

SOL Copy

```sol
16  constructor () {
17          _guardian = msg.sender;
18          emit GuardianPulled( address(0), _guardian );
19      }
20
21
22      function guardian() public view override returns (address) {
23          return _guardian;
24      }
25
26      modifier onlyGuardian() {
27          require( _guardian == msg.sender, "Guardable: caller is not the guard
28          _;
29      }
30
31      function renounceGuardian() public virtual override onlyGuardian() {
32          emit GuardianPushed( _guardian, address(0) );
33          _guardian = address(0);
34      }
35
36      function pushGuardian( address newGuardian_ ) public virtual override onl
37          require( newGuardian_ != address(0), "Guardable: new guardian is the
38          emit GuardianPushed( _guardian, newGuardian_ );
39           newGuardian = newGuardian ;
```

```
40      }
41
42      function pullGuardian() public virtual override {
43          require( msg.sender == _newGuardian, "Guardable: must be new guardian
44          emit GuardianPulled( _guardian, _newGuardian );
45          _guardian = _newGuardian;
46      }
47  }
```

Recommendation:

We advise it to do so as the latter `event` is utilized in functions like `pullGuardian` and the `constructor`.

Alleviation:

The `GuardianPulled` event is now properly emitted within the `renounceGuardian` function.

View Fix on GitHub

GUA-03M: Potentially Restrictive Functionality

Type	Severity	Location
Logical Fault	Minor	Guardable.sol:L37

Description:

The `pushGuardian` function validates that the `newGuardian_` being set is not the zero address, however, in doing so it will prevent a pending guardian to be overwritten with a zero entry.

Example:

contracts/types/Guardable.sol

SOL

Copy

```sol
function pushGuardian( address newGuardian_ ) public virtual override onlyGua
    require( newGuardian_ != address(0), "Guardable: new guardian is the zero
    emit GuardianPushed( _guardian, newGuardian_ );
    _newGuardian = newGuardian_;
}
```

Recommendation:

We advise this check to either be omitted entirely or only be evaluated when `_newGuardian` is itself equal to the zero-address as otherwise, an incorrectly set `_newGuardian` will not be able to be deleted.

Alleviation:

The restrictive `require` check is now omitted from the codebase.

View Fix on GitHub

PREV
GovernorAlpha.sol (GAA-M)

NEXT
OlympusTokenMigrator.sol (OTM-M)

OlympusTokenMigrator Manual Review Findings

ON THIS PAGE

OTM-01M: Improper Integration w/ Uniswap V2

Type	Severity	Location
Logical Fault	Major ●	OlympusTokenMigrator.sol:L266-L274, L281-L290

Description:

The way the migration of the token works can cause the migration to either completely halt or cause the liquidity position to significantly diminish in value should the governor address be a contract that can be actuated from anyone, such as a `Timelock` forked from Compound.

Example:

contracts/migration/OlympusTokenMigrator.sol

SOL

Copy

```sol
249 /**
250  *   @notice Migrate LP and pair with new OHM
251  */
252 function migrateLP(
253     address pair,
254     bool sushi,
```

```
255     address token
256 ) external onlyGovernor {
257     uint256 oldLPAmount = IERC20(pair).balanceOf(address(oldTreasury));
258     oldTreasury.manage(pair, oldLPAmount);
259 
260     IUniswapV2Router router = sushiRouter;
261     if (!sushi) {
262         router = uniRouter;
263     }
264 
265     IERC20(pair).approve(address(router), oldLPAmount);
266     (uint256 amountA, uint256 amountB) = router.removeLiquidity(
267         token,
268         address(oldOHM),
269         oldLPAmount,
270         0,
271         0,
272         address(this),
273         1000000000000
274     );
275 
276     newTreasury.mint(address(this), amountB);
277 
278     IERC20(token).approve(address(router), amountA);
279     newOHM.approve(address(router), amountB);
280 
281     router.addLiquidity(
282         token,
283         address(newOHM),
284         amountA,
285         amountB,
286         amountA,
287         amountB,
288         address(newTreasury),
289         100000000000
290     );
291 }
```

Recommendation:

We strongly recommend the migration procedure to be revised. In the current state, it specifies the expected output amounts as `0` which can cause an arbitreur to significantly skew the pair, diminish the LP position one-sidedly (i.e. towards OHM) and cause the liquidity removal to be in

the native token only. This can cause the protocol to crash due to the artificial inflation of OHM's price which can be performed with the help of flash loans if for example the governor is a `Timelock` implementation relying on a `GovernorAlpha` to actuate it. Additionally, the liquidity provision is also performed incorrectly as it specifies the amounts that should at minimum be set within the pair to be equal to the amounts provided. This case is only true when the pair has not been created before. Should `newOHM` units circulate in the market before this point, it would be possible for someone to race the transaction, create the pair with miniscule amounts and thus cause the migration to be impossible. As a last note, the current `block.timestamp` can and should be passed in as the expiry argument instead of the literal `1000000000000` which is meaningless.

Alleviation:

Minimum arguments were properly added to the `migrateLP` function and the numeric literal was substituted for the current `block.timestamp`, thereby alleviating this exhibit in full.

View Fix on GitHub

OTM-02M: Improper Evaluation of Token Balance

Type	Severity	Location
Logical Fault	Medium ●	OlympusTokenMigrator.sol:L364

Description:

The `_migrateToken` function is utilizing the `tokenValue` yielded by the `newTreasury` implementation yet is comparing it with the `excessReserves` of the `oldTreasury`.

Example:

contracts/migration/OlympusTokenMigrator.sol

SOL

Copy

```
357 /**
358  *   @notice Migrate token from old treasury to new treasury
359  */
360 function _migrateToken(address token, bool deposit) internal {
361     uint256 balance = IERC20(token).balanceOf(address(oldTreasury));
362 
363     uint256 excessReserves = oldTreasury.excessReserves();
364     uint256 tokenValue = newTreasury.tokenValue(token, balance);
365 
366     if (tokenValue > excessReserves) {
367         tokenValue = excessReserves;
368         balance = excessReserves * 10**9;
369     }
370 
371     oldTreasury.manage(token, balance);
372 
373     if (deposit) {
374         IERC20(token).safeApprove(address(newTreasury), balance);
375         newTreasury.deposit(balance, token, tokenValue);
376     } else {
377         IERC20(token).transfer(address(newTreasury), balance);
378     }
379 }
```

Recommendation:

We strongly recommend the `valueOf` implementation of the legacy treasury to be utilized instead as it is currently incorrectly evaluating the maximum value that can be retrieved from `oldTreasury`.

Alleviation:

The code now properly uttilizes the legacy `valueOf` function to properly identify how many funds can be managed from the legacy treasury.

View Fix on GitHub

OTM-03M: Ungraceful Mint Handling

Type	Severity	Location
Logical Fault	Medium	OlympusTokenMigrator.sol:L217, L219

Description:

The contract contains logic blocks that indicate the possibility of `oldOHM` being minted beyond migration is realistic and simply prohibits swaps for it, however, a mint event and corresponding transfer to the migrator contract is unaccounted for and can cause `defund` to be inoperable.

Example:

contracts/migration/OlympusTokenMigrator.sol

SOL

Copy

```sol
208 // withdraw backing of migrated OHM
209 function defund(address reserve) external onlyGovernor {
210     require(ohmMigrated && timelockEnd < block.number && timelockEnd != 0);
211     oldwsOHM.unwrap(oldwsOHM.balanceOf(address(this)));
212
213     uint256 amountToUnstake = oldsOHM.balanceOf(address(this));
214     oldsOHM.approve(address(oldStaking), amountToUnstake);
215     oldStaking.unstake(amountToUnstake, false);
216
217     uint256 balance = oldOHM.balanceOf(address(this));
218
219     oldSupply = oldSupply.sub(balance);
220
221     uint256 amountToWithdraw = balance.mul(1e9);
222     oldOHM.approve(address(oldTreasury), amountToWithdraw);
223     oldTreasury.withdraw(amountToWithdraw, reserve);
224     IERC20(reserve).safeTransfer(address(newTreasury), IERC20(reserve).balanc
225
226     emit Defunded(balance);
227 }
```

Recommendation:

We strongly recommend the code to gracefully handle such an event by containing an `if` block that nullifies `oldSupply` if the `balance` exceeds it.

Alleviation:

The code now properly handles an instance of the `balance` exceeding the `oldSupply` in accordance to our recommendation.

View Fix on GitHub

OTM-04M: Potential of Repeat Invocation

Type	Severity	Location
Logical Fault	Minor	OlympusTokenMigrator.sol:L229-L234

Description:

The `startTimelock` function can be invoked multiple times.

Example:

contracts/migration/OlympusTokenMigrator.sol

SOL

Copy

```sol
229 // start timelock to send backing to new treasury
230 function startTimelock() external onlyGovernor {
231     timelockEnd = block.number.add(timelockLength);
232
233     emit TimelockStarted(block.number, timelockEnd);
234 }
```

Recommendation:

It should only be invoke-able once and as such should introduce a `require` check that ensures `timelockEnd` is equal to `0` at the beginning.

Alleviation:

The `require` check we recommended was properly added to the codebase.

View Fix on GitHub

PREV
Guardable.sol (GUA-M)

NEXT
Ownable.sol (OWN-M)

Ownable Manual Review Findings

ON THIS PAGE

OWN-01M: Improper Ownership Renouncation

Type	Severity	Location
Logical Fault	Medium	Ownable.sol:L31-L34

Description:

The `renounceManagement` function deletes the current `_owner` in place, however, the `_newOwner` remains especially so when the `pullManagement` function is invoked once as it does not delete the previous entry. This would permit ownership to be re-instated even after it has been renounced on a particular contract.

Example:

contracts/types/Ownable.sol

SOL

Copy

```sol
function renounceManagement() public virtual override onlyOwner() {
    emit OwnershipPushed( _owner, address(0) );
    _owner = address(0);
}
```

Recommendation:

We strongly recommend the `_newOwner` to also be deleted when `renounceManagement` is invoked to prevent improper state transitions.

Alleviation:

The new owner is properly deleted when ownership is renounced.

View Fix on GitHub

OWN-02M: Incorrect Events Emitted

Type	Severity	Location
Logical Fault	Minor ●	Ownable.sol:L16

Description:

The `constructor` and `renounceManagement` function of the `Ownable` contract incorrectly emit the `OwnershipPushed` event instead of the `OwnershipPulled` one.

Example:

contracts/types/Ownable.sol

SOL Copy

```sol
11  event OwnershipPushed(address indexed previousOwner, address indexed newOwner
12  event OwnershipPulled(address indexed previousOwner, address indexed newOwner
13
14  constructor () {
15      _owner = msg.sender;
16      emit OwnershipPushed( address(0), _owner );
17  }
18
19  function owner() public view override returns (address) {
20      return _owner;
21  }
22
23  modifier onlyOwner() {
24      require( _owner == msg.sender, "Ownable: caller is not the owner" );
25      _;
26  }
27
28  function renounceManagement() public virtual override onlyOwner() {
29      emit OwnershipPushed( _owner, address(0) );
30      _owner = address(0);
31  }
32
33  function pushManagement( address newOwner_ ) public virtual override onlyOwne
34      require( newOwner_ != address(0), "Ownable: new owner is the zero address
```

```
35      emit OwnershipPushed( _owner, newOwner_ );
36      _newOwner = newOwner_;
37  }
38
39  function pullManagement() public virtual override {
40      require( msg.sender == _newOwner, "Ownable: must be new owner to pull");
41      emit OwnershipPulled( _owner, _newOwner );
42      _owner = _newOwner;
43  }
```

Recommendation:

We advise the latter to be utilized as it is the canonical one when an `_owner` entry is written in `pullManagement`.

Alleviation:

The `OwnershipPulled` event is now properly emitted within the `renounceOwnership` function.

View Fix on GitHub

OWN-03M: Potentially Restrictive Functionality

Type	Severity	Location
Logical Fault	Minor ●	Ownable.sol:L34

Description:

The `pushManagement` function validates that the `newOwner_` being set is not the zero address, however, in doing so it will prevent a pending owner to be overwritten with a zero entry.

Example:

contracts/types/Ownable.sol

SOL

Copy

```sol
33  function pushManagement( address newOwner_ ) public virtual override onlyOwne
34      require( newOwner_ != address(0), "Ownable: new owner is the zero address
35      emit OwnershipPushed( _owner, newOwner_ );
36      _newOwner = newOwner_;
37  }
```

Recommendation:

We advise this check to either be omitted entirely or only be evaluated when `_newOwner` is itself equal to the zero-address as otherwise, an incorrectly set `_newOwner` will not be able to be deleted.

Alleviation:

The restrictive `require` check is now omitted from the codebase.

View Fix on GitHub

PREV
OlympusTokenMigrator.sol (OTM-M)

NEXT
StakingDistributor.sol (SDR-M)

StakingDistributor Manual Review Findings

ON THIS PAGE

SDR-01M: Improper Accumulation of Rewards

Type	Severity	Location
Logical Fault	Medium	StakingDistributor.sol:L110-L123

Description:

The `nextRewardFor` function does not properly accumulate rewards if multiple ones are specified for a particular recipient.

Example:

contracts/StakingDistributor.sol

SOL

Copy

```sol
110 /**
111     @notice view function for next reward for specified address
112     @param _recipient address
113     @return uint
114  */
115 function nextRewardFor(address _recipient) public view returns (uint256) {
116     uint256 reward;
117     for (uint256 i = 0; i < info.length; i++) {
```

```
118         if (info[i].recipient == _recipient) {
119             reward = nextRewardAt(info[i].rate);
120         }
121     }
122     return reward;
123 }
```

Recommendation:

We advise the function to properly sum the results of `nextRewardAt` invocations to ensure it operates as intended.

Alleviation:

Rewards are now properly accumulated in the `reward` entry.

View Fix on GitHub

SDR-02M: Ungraceful Handling of High Adjustment Rates

Type	Severity	Location
Mathematical Operations	Medium ●	StakingDistributor.sol:L90

Description:

The `adjustment.rate` is meant to represent a step-by-step reduction or increase of the reward rate for a particular recipient, however, there can be a case where the `info[_index].rate` is smaller than the step which would render the `adjust` operation impossible and thus cause the full `distribute` hook to fail.

Example:

contracts/StakingDistributor.sol

SOL

Copy

```sol
88  } else {
89      // if rate should decrease
90      info[_index].rate = info[_index].rate.sub(adjustment.rate); // lower rate
91      if (info[_index].rate <= adjustment.target) {
92          // if target met
93          adjustments[_index].rate = 0; // turn off adjustment
94      }
95  }
```

Recommendation:

We advise the reduction of a particular rate to be gracefully handled whereby if the reduction is greater than the current rate the rate should be set to zero.

Alleviation:

The Olympus DAO team considered this exhibit but decided to retain the current behaviour of the code in place.

SDR-03M: Inexistent Validation of Entry Validity

Type	Severity	Location
Input Sanitization	Minor •	StakingDistributor.sol:L142-L147, L156-L169

Description:

The `removeRecipient` and `setAdjustment` functions do not actually validate whether there is an existing entry in the `_index` they are operating in, leading to adjustments for inexistent entries / future ones or removal of inexistent entries.

Example:

contracts/StakingDistributor.sol

SOL

Copy

```sol
137 /**
138     @notice removes recipient for distributions
139     @param _index uint
140     @param _recipient address
141  */
142 function removeRecipient(uint256 _index, address _recipient) external {
143     require(msg.sender == governor() || msg.sender == guardian(), "Caller is
144     require(_recipient == info[_index].recipient);
145     info[_index].recipient = address(0);
146     info[_index].rate = 0;
147 }
148
149 /**
150     @notice set adjustment info for a collector's reward rate
151     @param _index uint
152     @param _add bool
153     @param _rate uint
154     @param _target uint
155  */
156 function setAdjustment(
157     uint256 _index,
158     bool _add,
```

```
159     uint256 _rate,
160     uint256 _target
161 ) external {
162     require(msg.sender == governor() || msg.sender == guardian(), "Caller is
163
164     if (msg.sender == guardian()) {
165         require(_rate <= info[_index].rate.mul(25).div(1000), "Limiter: canno
166     }
167
168     adjustments[_index] = Adjust({add: _add, rate: _rate, target: _target});
169 }
```

Recommendation:

We advise both functions to properly validate that an `info` entry exists by evaluating its `recipient`.

Alleviation:

Both functions now properly validate the existance of an entry by ensuring that the `info[_index].recipient` member is non-zero.

View Fix on GitHub

SDR-04M: Inexistent Validation of Reward Rate

Type	Severity	Location
Input Sanitization	Minor •	StakingDistributor.sol:L134

Description:

The `addRecipient` function does not validate that the `_rewardRate` set does not exceed the maximum achievable which is `1000000`.

Example:

contracts/StakingDistributor.sol

SOL

Copy

```sol
127 /**
128     @notice adds recipient for distributions
129     @param _recipient address
130     @param _rewardRate uint
131  */
132 function addRecipient(address _recipient, uint256 _rewardRate) external onlyG
133     require(_recipient != address(0));
134     info.push(Info({recipient: _recipient, rate: _rewardRate}));
135 }
```

Recommendation:

We advise such validation to be imposed to prevent arbitrarily high reward rates.

Alleviation:

The function now properly validate that the `_rewardRate` set is at most equivalent to the newly declared `rateDenominator`.

View Fix on GitHub

PREV
Ownable.sol (OWN-M)

NEXT
StandardBondingCalculator.sol (SBC-M)

StandardBondingCalculator Manual Review Findings

ON THIS PAGE

SBC-01M: Inexistent Validation of Pair Tokens

Type	Severity	Location
Logical Fault	Major	StandardBondingCalculator.sol:L50-L60

Description:

The `markdown` function incorrectly assumes that if the `token0` of a pair is not the `OHM` address, `token1` will be so which may not be the case.

Example:

contracts/StandardBondingCalculator.sol

SOL

Copy

```sol
50  function markdown( address _pair ) external view override returns ( uint ) {
51      ( uint reserve0, uint reserve1, ) = IUniswapV2Pair( _pair ).getReserves()
52
53      uint reserve;
54      if ( IUniswapV2Pair( _pair ).token0() == address( OHM ) ) {
55          reserve = reserve1;
56      } else {
57          reserve = reserve0;
58      }
59      return reserve.mul( 2 * ( 10 ** IERC20Metadata(address(OHM)).decimals() )
```

```
60  }
```

Recommendation:

We advise a `require` check to be introduced in the `else` chain of the `if` clause that mandates `token1` to be the `OHM` address.

Alleviation:

A `require` check was introduced in the `else` case that mandates `token1` to be equivalent to `OHM` thereby alleviating this exhibit.

View Fix on GitHub

SBC-02M: Incorrect Usage of `SafeMath` Library

Type	Severity	Location
Language Specific	Minor ●	StandardBondingCalculator.sol:L5, L21

Description:

The `using SafeMath for uint112` statement is ineffectual as all `SafeMath` operations that will be performed on the `uint112` data type will indirectly cast the value to a `uint256` and yield the `uint256` result which if casted to a `uint112` can still overflow.

Example:

contracts/StandardBondingCalculator.sol

SOL

Copy

```
21  using SafeMath for uint112;
```

Recommendation:

We advise either the `SafeMath` library implementation to be expanded to support the `uint112` data type or the `using` statement to be omitted should it be considered unnecessary in the codebase and replaced by `uint256` casts to `uint112` variables that are used in these calculations.

Alleviation:

The ineffectual `using * for` statement was omitted from the codebase.

View Fix on GitHub

PREV
StakingDistributor.sol (SDR-M)

NEXT

Treasury Manual Review Findings

ON THIS PAGE

TRE-01M: Insecure Management of Reserve & Liquidity Tokens

Type	Severity	Location
Logical Fault	Major ●	Treasury.sol:L242, L244, L246, L248, L270, L333

Description:

The `registry` arrays of reserve and liquidity tokens are improperly maintained which can cause huge discrepancies to the `totalReserves` measured in the `auditReserves` function. As an example, duplicate entries within a single array can cause a particular reserve to be calculated twice whereas an asset being a reserve and liquidity token at the same time will also cause an incorrect duplicate measurement of its balance value.

Example:

contracts/Treasury.sol

SOL

Copy

```sol
254 /**
255  * @notice enable permission from queue
256  * @param _status STATUS
```

```
257  * @param _address address
258  * @param _calculator address
259  */
260 function enable(
261     STATUS _status,
262     address _address,
263     address _calculator
264 ) external onlyOwner {
265     require(onChainGoverned, "OCG Not Enabled: Use queueTimelock");
266     if (_status == STATUS.SOHM) {
267         // 9
268         sOHM = IERC20(_address);
269     } else {
270         registry[_status].push(_address);
271         permissions[_status][_address] = true;
272 
273         if (_status == STATUS.LIQUIDITYTOKEN) {
274             // 5
275             bondCalculator[_address] = _calculator;
276         }
277     }
278     emit Permissioned(_address, _status, true);
279 }
```

Recommendation:

We advise all actions modifying those arrays to properly check for duplicates by preventing re-setting the same permission for an address to `true`.

Alleviation:

Both `registry` adjustment code blocks were updated to properly evaluate whether duplicate entries exist as well as to delete any previously set state in case the same token is being set between a liquidity and reserve token and vice versa.

View Fix on GitHub

TRE-02M: Weak Debt Position Validation

Type	Severity	Location
Logical Fault	Major ●	Treasury.sol:L140-L163

Description:

The `incurDebt` function mandates that the caller has a sufficient balance of `sOHM` to create the debt position, however, `sOHM` is a freely transferrable asset that should only be used as a data point when it cannot be transferred.

Example:

contracts/Treasury.sol

SOL Copy

```sol
/**
    @notice allow approved address to borrow reserves
    @param _amount uint
    @param _token address
 */
function incurDebt(uint256 _amount, address _token) external override {
    require(permissions[STATUS.DEBTOR][msg.sender], "Not approved");
    require(permissions[STATUS.RESERVETOKEN][_token], "Not accepted");

    uint256 value = tokenValue(_token, _amount);
    require(value != 0);

    uint256 availableDebt = sOHM.balanceOf(msg.sender).sub(debtorBalance[msg.
    require(value <= availableDebt, "Exceeds debt limit");

    debtorBalance[msg.sender] = debtorBalance[msg.sender].add(value);
    totalDebt = totalDebt.add(value);

    totalReserves = totalReserves.sub(value);

    IERC20(_token).transfer(msg.sender, _amount);

    emit CreateDebt(msg.sender, _token, _amount, value);
```

```
162     emit CreateDebt(msg.sender, _token, _amount, value);
163 }
```

Recommendation:

We advise the `sOHM` to either be held in custody or for some other similar mechanism to be put in place as the current debt mechanism is circumventable.

Alleviation:

The debt management system has now been built-in the `sOHM` implementation which in turn prevents transfers that would reduce the holder's balance below the required debt threshold.

View Fix on GitHub

TRE-03M: Improperly Valid Case

Type	Severity	Location
Mathematical Operations	Medium	Treasury.sol:L150, L380-L386

Description:

The `tokenValue` function should not yield a value of `0` under any circumstances as it will result in no-ops when utilized in mathematical operations and can cause the system to misbehave in case i.e. a token has more decimals than OHM is utilized in the evaluation that is not a `LIQUIDITYTOKEN`.

Example:

contracts/Treasury.sol

SOL

Copy

```
374 /**
375     @notice returns OHM valuation of asset
376     @param _token address
377     @param _amount uint
378     @return value_ uint
379  */
380 function tokenValue(address _token, uint256 _amount) public view override ret
381     value_ = _amount.mul(10**IERC20Metadata(address(OHM)).decimals()).div(10*
382
383     if (permissions[STATUS.LIQUIDITYTOKEN][_token]) {
384         value_ = IBondingCalculator(bondCalculator[_token]).valuation(_token,
385     }
386 }
```

Recommendation:

We strongly recommend the first linked `require` check to be relocated to the `tokenValue` function itself as no zero evaluations should be considered "valid".

Alleviation:

Alleviation:

The Olympus DAO team considered this exhibit but decided to retain the current behaviour of the code in place.

TRE-04M: Inexistent Validation of Token Status

Type	Severity	Location
Logical Fault	Medium	Treasury.sol:L208-L210

Description:

The `manage` function does not properly validate the permission status of the input `_token` which can improperly manipulate the `totalReserves` i.e. with no-op function implementations.

Example:

contracts/Treasury.sol

SOL

Copy

```sol
/**
    @notice allow approved address to withdraw assets
    @param _token address
    @param _amount uint
 */
function manage(address _token, uint256 _amount) external override {
    if (permissions[STATUS.LIQUIDITYTOKEN][_token]) {
        require(permissions[STATUS.LIQUIDITYMANAGER][msg.sender], "Not approv
    } else {
        require(permissions[STATUS.RESERVEMANAGER][msg.sender], "Not approved
    }

    uint256 value = tokenValue(_token, _amount);
    require(value <= excessReserves(), "Insufficient reserves");

    totalReserves = totalReserves.sub(value);

    IERC20(_token).safeTransfer(msg.sender, _amount);

    emit ReservesManaged(_token, _amount);
}
```

Recommendation:

Recommendation:

We advise the `else` branch of the first clause in `manage` to impose a `require` check ensuring that the permission of the `_token` specified is as a `RESERVETOKEN`.

Alleviation:

The `totalReserves` value is now adjusted solely when the input `_token` falls under either the liquidity or reserve token category, thereby alleviating this exhibit.

View Fix on GitHub

TRE-05M: Potentially Unsafe Primitive Evaluation

Type	Severity	Location
Standard Conformity	Minor ●	Treasury.sol:L380-L386

Description:

The evaluation of a particular token's value when the token is not a `LIQUIDITYTOKEN` is performed by a simple decimal-based conversion.

Example:

contracts/Treasury.sol

SOL

Copy

```
374 /**
375     @notice returns OHM valuation of asset
376     @param _token address
377     @param _amount uint
378     @return value_ uint
379  */
380 function tokenValue(address _token, uint256 _amount) public view override ret
381     value_ = _amount.mul(10**IERC20Metadata(address(OHM)).decimals()).div(10*
382
383     if (permissions[STATUS.LIQUIDITYTOKEN][_token]) {
384         value_ = IBondingCalculator(bondCalculator[_token]).valuation(_token,
385     }
386 }
```

Recommendation:

Apart from considering these tokens equal in value, it also directly relies on the presence of the `decimals` operator on the token which at times may not be available as it is an OPTIONAL member of the **EIP-20** standard. We advise this particular trait to be considered carefully in the overall system as checks can be imposed at the inclusion level to prevent non-compliant tokens from being added.

Alleviation:

The Olympus DAO team considered this exhibit but decided to retain the current behaviour of the code in place.

PREV
StandardBondingCalculator.sol (SBC-M)

NEXT
VaultOwned.sol (VOD-M)

VaultOwned Manual Review Findings

ON THIS PAGE

VOD-01M: Centralized Sensitive Functionality

Type	Severity	Location
Logical Fault	Medium	VaultOwned.sol:L10-L14

Description:

The `setVault` function sets the current `_vault` in place for `onlyVault` modifier enforcement, however, it can be invoked an arbitrary number of times, can override the current `_vault` in place and does not contain an `override` keyword meaning that it is meant to be invoked by EOAs or similar actors.

Example:

contracts/types/VaultOwned.sol

SOL

Copy

```sol
10  function setVault( address vault_ ) external onlyOwner() returns ( bool ) {
11    _vault = vault_;
12
13    return true;
14  }
```

Recommendation:

We advise it to potentially disallow over-writing the `_vault` in place once it has been set once as otherwise, it can become a single point of failure for the system. Additionally, we advise the

returned `bool` to be omitted given that it is always `true` and is a non-standard function.

Alleviation:

The Olympus DAO team considered this exhibit but decided to retain the current behaviour of the code in place.

PREV
Treasury.sol (TRE-M)

NEXT
gOHM.sol (OHM-M)

gOHM Manual Review Findings

ON THIS PAGE

OHM-01M: Improper State Control of Migration

Type	Severity	Location
Logical Fault	Medium	gOHM.sol:L120-L134

Description:

The `migrate` function, as its documentation states, should only be invoke-able once during the contract migration, however, the logical checks it enforces allow it to be invoked and thus set very sensitive contract variables an arbitrary number of times.

Example:

contracts/governance/gOHM.sol

SOL

Copy

```sol
120 /**
121  * @notice transfer mint rights from migrator to staking
122  * @notice can only be done once, at the time of contract migration
123  * @param _staking address
124  * @param _sOHM address
125  */
126 function migrate(address _staking, address _sOHM) external override onlyAppro
127     require(_staking != approved);
128
129     require(_staking != address(0));
130     approved = _staking;
131
```

```
132     require(_sOHM != address(0));
133     sOHM = IsOHM(_sOHM);
134 }
```

Recommendation:

We strongly recommend the function to ensure that `sOHM` has not been previously set as otherwise, there is no protection preventing re-setting those variables.

Alleviation:

A dedicated `migrated` flag has been introduced to the codebase and is being used as a flag to indicate whether `migrate` has been invoked before thereby preventing its re-execution and alleviating this exhibit.

View Fix on GitHub

PREV
VaultOwned.sol (VOD-M)

NEXT
sOlympusERC20.sol (OEC-M)

sOlympusERC20 Manual Review Findings

ON THIS PAGE

OEC-01M: Potentially Incorrect Extrapolation of Rebase

Type	Severity	Location
Logical Fault	Medium	sOlympusERC20.sol:L122

Description:

The `rebase` function will extrapolate the `rebaseAmount` should a non-zero amount of `circulatingSupply` be returned as the calculation performed multiplies the profit by the `_totalSupply` and divides it by the `circulatingSupply_`, the latter of which is guaranteed to be greater than the former thus causing the `profit_` to be increased.

Example:

contracts/sOlympusERC20.sol

SOL

Copy

```sol
108 /**
109     @notice increases rOHM supply to increase staking balances relative to pr
110     @param profit_ uint256
111     @return uint256
112  */
113 function rebase( uint256 profit_, uint epoch_ ) public onlyStakingContract()
114     uint256 rebaseAmount;
115     uint256 circulatingSupply_ = circulatingSupply();
116
117     if ( profit_ == 0 ) {
118         emit LogSupply( epoch , block.timestamp,  totalSupply );
```

```
119         emit LogRebase( epoch_, 0, index() );
120         return _totalSupply;
121     } else if ( circulatingSupply_ > 0 ){
122         rebaseAmount = profit_.mul( _totalSupply ).div( circulatingSupply_ );
123     } else {
124         rebaseAmount = profit_;
125     }
126
127     _totalSupply = _totalSupply.add( rebaseAmount );
128
129     if ( _totalSupply > MAX_SUPPLY ) {
130         _totalSupply = MAX_SUPPLY;
131     }
132
133     _gonsPerFragment = TOTAL_GONS.div( _totalSupply );
134
135     _storeRebase( circulatingSupply_, profit_, epoch_ );
136
137     return _totalSupply;
138 }
```

Recommendation:

We advise this trait to be carefully assessed and if desired to be properly documented as it can cause disproportionate profits to be calculated.

Alleviation:

The Olympus DAO team considered this exhibit, identified it as desired behaviour but opted not to apply any remediation for it.

PREV
gOHM.sol (OHM-M)

NEXT
BondDepository.sol (BDY-C)

ID	Severity	Addressed	Title
BDY-03M	Medium	No	Inexplicable Optional Value of Decay
BTR-01M	Major	Yes	Confusion of Value Denominations
BTR-02M	Medium	No	Artificial Inflation Mechanism
BTR-03M	Minor	Yes	Inexistent Redemption of FEO Fees
BTR-04M	Minor	No	Inexistent Validation of Non-Zero Redemption
ERC-01M	Minor	Yes	Non-Standard Mint Implementation
ERP-01M	Medium	Yes	Insecure Elliptic Curve Recovery Mechanism
ERP-02M	Minor	Yes	Cross-Chain Signature Replay Attack Susceptibility
FPT-01M	Major	No	Potentially Invalid Implementation
GOV-01M	Medium	Yes	Improper Governor Renouncation
GOV-02M	Minor	Yes	Incorrect Event Emitted
GOV-03M	Minor	Yes	Potentially Restrictive Functionality
GAA-01M	Informational	Yes	Improper Percentage Documented
GUA-01M	Medium	Yes	Improper Guardian Renouncation
GUA-02M	Minor	Yes	Incorrect Event Emitted
GUA-03M	Minor	Yes	Potentially Restrictive Functionality
OTM-01M	Major	Yes	Improper Integration w/ Uniswap V2
OTM-02M	Medium	Yes	Improper Evaluation of Token Balance
OTM-03M	Medium	Yes	Ungraceful Mint Handling

ID	Severity	Addressed	Title
OTM-03M	Medium	Yes	Ungraceful Mint Handling
OTM-04M	Minor	Yes	Potential of Repeat Invocation
OWN-01M	Medium	Yes	Improper Ownership Renouncation
OWN-02M	Minor	Yes	Incorrect Events Emitted
OWN-03M	Minor	Yes	Potentially Restrictive Functionality
SDR-01M	Medium	Yes	Improper Accumulation of Rewards
SDR-02M	Medium	No	Ungraceful Handling of High Adjustment Rates
SDR-03M	Minor	Yes	Inexistent Validation of Entry Validity
SDR-04M	Minor	Yes	Inexistent Validation of Reward Rate
SBC-01M	Major	Yes	Inexistent Validation of Pair Tokens
SBC-02M	Minor	Yes	Incorrect Usage of `SafeMath` Library
TRE-01M	Major	Yes	Insecure Management of Reserve & Liquidity Tokens
TRE-02M	Major	Yes	Weak Debt Position Validation
TRE-03M	Medium	No	Improperly Valid Case
TRE-04M	Medium	Yes	Inexistent Validation of Token Status
TRE-05M	Minor	No	Potentially Unsafe Primitive Evaluation
VOD-01M	Medium	No	Centralized Sensitive Functionality
OHM-01M	Medium	Yes	Improper State Control of Migration
OEC-01M	Medium	No	Potentially Incorrect Extrapolation of Rebase

PREV
Static Analysis

NEXT
Code Style

Code Style

During the manual portion of the audit, we identified **34 optimizations** that can be applied to the codebase that will decrease the gas-cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
BDY-01C	Informational	No	Improper Error Messages
BDY-02C	Informational	No	Inexistent Error Messages
BDY-03C	Informational	No	Potentially Misleading USD Evaluation
BDY-04C	Informational	No	Redundant Explicit Zero-Value Assignment
BTR-01C	Informational	Yes	Inexistent Variable Visibility Specifiers
ERC-01C	Informational	Yes	Variable Mutability Specifier
ERP-01C	Informational	Yes	Improper Error Name
ERP-02C	Informational	Yes	Redundant Visibility Specifiers
ERP-03C	Informational	Yes	Suboptimal Code Style
FMH-01C	Informational	Yes	Outdated Implementation
GOV-01C	Informational	Yes	Inexistent Deletion of Pending Governor

BondDepository Code Style Findings

ON THIS PAGE

BDY-01C: Improper Error Messages

Type	Severity	Location
Code Style	Informational ●	BondDepository.sol:L187, L191

Description:

The linked error messages state that the bond has concluded, however, the same error will arise if the `capacity` is simply exceeded rather than `0` and would otherwise succeed with a `payout` / `_amount` less than the limit.

Example:

contracts/BondDepository.sol

SOL

Copy

```sol
184 // ensure there is remaining capacity for bond
185 if (info.capacityIsPayout) {
186   // capacity in payout terms
187   require(info.capacity >= payout, "Bond concluded");
188   info.capacity = info.capacity.sub(payout);
189 } else {
190   // capacity in principal terms
191   require(info.capacity >= _amount, "Bond concluded");
```

```
192   info.capacity = info.capacity.sub(_amount);
193 }
```

Recommendation:

We advise the error message to be more descriptive to aid users in adjusting the variables correctly to ensure their action succeeds.

Alleviation:

The Olympus DAO team considered this exhibit but opted not to apply any remediation for it.

BDY-02C: Inexistent Error Messages

Type	Severity	Location
Code Style	Informational ●	BondDepository.sol:L68, L70, L144, L145

Description:

The linked `require` checks contain no descriptive error messages.

Example:

contracts/BondDepository.sol

SOL

Copy

```sol
67  constructor(address _OHM, address _treasury) {
68    require(_OHM != address(0));
69    OHM = IERC20(_OHM);
70    require(_treasury != address(0));
71    treasury = ITreasury(_treasury);
72  }
```

Recommendation:

We advise them to be set so to aid in the debugging of the application and to also enable more accurate validation of the `require` condition purposes.

Alleviation:

The Olympus DAO team considered this exhibit but opted not to apply any remediation for it.

BDY-03C: Potentially Misleading USD Evaluation

Type	Severity	Location
Code Style	Informational	BondDepository.sol:L343-L355

Description:

The on-chain USD evaluation of a bond is solely used for events, however, it appears to be incorrect as its comments indicate that a DAI value is calculated whereas the principal's paired assets are simply used with no relation to the actual DAI token.

Example:

contracts/BondDepository.sol

SOL

Copy

```sol
343 /**
344  * @notice converts bond price to DAI value
345  * @param _BID uint
346  * @return price_ uint
347  */
348 function bondPriceInUSD(uint256 _BID) public view returns (uint256 price_) {
349   Bond memory bond = bonds[_BID];
350   if (address(bond.calculator) != address(0)) {
351     price_ = bondPrice(_BID).mul(bond.calculator.markdown(address(bond.princi
352   } else {
353     price_ = bondPrice(_BID).mul(10**IERC20Metadata(address(bond.principal)).
354   }
355 }
```

Recommendation:

We advise either the code or the comments surrounding it to be revised to better illustrate the function's purpose.

Alleviation:

The Olympus DAO team considered this exhibit but opted not to apply any remediation for it.

BDY-04C: Redundant Explicit Zero-Value Assignment

Type	Severity	Location
Gas Optimization	Informational ●	BondDepository.sol:L89

Description:

The `addBond` function redundantly assigns the `Terms memory terms` pointer to an uninitialized `Terms` entry.

Example:

contracts/BondDepository.sol

SOL

Copy

```sol
/**
 * @notice creates a new bond type
 * @param _principal address
 * @param _calculator address
 * @param _capacity uint
 * @param _capacityIsPayout bool
 */
function addBond(
  address _principal,
  address _calculator,
  uint256 _capacity,
  bool _capacityIsPayout
) external onlyGuardian returns (uint256 id_) {
  Terms memory terms = Terms({controlVariable: 0, fixedTerm: false, vestingTe

  bonds[IDs.length] = Bond({principal: IERC20(_principal), calculator: IBondi

  id_ = IDs.length;
  IDs.push(_principal);
}
```

Recommendation:

We advise the assignment to be omitted and the pointer to be used in the next statement directly as it points to an uninitialized `Terms` struct when declared.

Alleviation:

The Olympus DAO team considered this exhibit but opted not to apply any remediation for it.

PREV
sOlympusERC20.sol (OEC-M)

NEXT
BondTeller.sol (BTR-C)

BondTeller Code Style Findings

ON THIS PAGE

BTR-01C: Inexistent Variable Visibility Specifiers

Type	Severity	Location
Code Style	Informational	BondTeller.sol:L44-L49

Description:

The linked variables have no visibility specifier explicitly set.

Example:

contracts/BondTeller.sol

SOL

Copy

```sol
address depository; // contract where users deposit bonds
IStaking immutable staking; // contract to stake payout
ITreasury immutable treasury;
IERC20 immutable OHM;
IERC20 immutable sOHM; // payment token
IgOHM immutable gOHM;
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current compiler behaviour is to assign a specifier automatically.

Alleviation:

All variables were set to `internal` thereby alleviating this exhibit.

View Fix on GitHub

PREV
BondDepository.sol (BDY-C)

NEXT
ERC20.sol (ERC-C)

ERC20 Code Style Findings

ON THIS PAGE

ERC-01C: Variable Mutability Specifier

Type	Severity	Location
Gas Optimization	Informational	ERC20.sol:L26, L31

Description:

As the `ERC20` token implementation is used internally, the `_decimals` member is only set once during the `constructor` of the contract.

Example:

contracts/types/ERC20.sol

SOL

Copy

```sol
uint8 internal _decimals;

constructor (string memory name_, string memory symbol_, uint8 decimals_) {
    _name = name_;
    _symbol = symbol_;
    _decimals = decimals_;
}
```

Recommendation:

We advise it to be set as `immutable` greatly opimizing its read access gas cost.

Alleviation:

The `_decimals` member was properly set as `immutable`.

View Fix on GitHub

PREV
BondTeller.sol (BTR-C)

NEXT
ERC20Permit.sol (ERP-C)

ERC20Permit Code Style Findings

ON THIS PAGE

ERP-01C: Improper Error Name

Type	Severity	Location
Code Style	Informational	ERC20Permit.sol:L56

Description:

The error name references another contract that does not exist in the codebase.

Example:

contracts/types/ERC20Permit.sol

SOL

Copy

```
56  require(signer != address(0) && signer == owner, "ZeroSwapPermit: Invalid sig
```

Recommendation:

We advise the error to be renamed.

Alleviation:

The error message has been corrected to utilize the contract's name instead.

View Fix on GitHub

ERP-02C: Redundant Visibility Specifiers

Type	Severity	Location
Gas Optimization	Informational	ERC20Permit.sol:L15, L17

Description:

The linked variables are meant to be internally available `constant` variables.

Example:

contracts/types/ERC20Permit.sol

SOL

Copy

```sol
14  // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonc
15  bytes32 public constant PERMIT_TYPEHASH = 0x6e71edae12b1b97f4d1f60370fef10105
16
17  bytes32 public DOMAIN_SEPARATOR;
```

Recommendation:

We advise them to be set as `private` or `internal` to reduce the codebase bloat and bytecode size of the contract.

Alleviation:

Visibility specifiers were adjusted for both variables to ensure they are no longer exposed publicly.

View Fix on GitHub

ERP-03C: Suboptimal Code Style

Type	Severity	Location
Code Style	Informational	ERC20Permit.sol:L14, L15

Description:

The `PERMIT_TYPEHASH` variable has the hash declared as its assignment and the `keccak256` instruction as its comment.

Example:

contracts/types/ERC20Permit.sol

SOL

Copy

```
14  // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonc
15  bytes32 public constant PERMIT_TYPEHASH = 0x6e71edae12b1b97f4d1f60370fef10105
```

Recommendation:

We advise these to be reversed as the one-way association of `keccak256` is not guaranteed from a hash to its value but rather from a value to its hash. Additionally, validation of the values would have less implications as the `keccak256` instruction is guaranteed to be performed properly. To note, assigning a `constant` to a `keccak256` instruction's result does not affect gas cost as the value is pre-calculated by the compiler.

Alleviation:

The typehash is now properly assigned to the evaluation of a `keccak256` instruction and is set as `immutable` to benefit from the gas optimization of evaluating the expression once.

View Fix on GitHub

PREV
ERC20.sol (ERC-C)

NEXT
FullMath.sol (FMH-C)

FullMath Code Style Findings

ON THIS PAGE

FMH-01C: Outdated Implementation

Type	Severity	Location
Gas Optimization	Informational	FullMath.sol:L33-L44

Description:

The `FullMath` contract is a copy of the homonymous Uniswap V2 library and implements an outdated version of the `mulDiv` function which does not contain the **optimization of** `h == 0` as present in the Uniswap V2 equivalent.

Example:

contracts/libraries/FullMath.sol

SOL

Copy

```sol
function mulDiv(
    uint256 x,
    uint256 y,
    uint256 d
) internal pure returns (uint256) {
    (uint256 l, uint256 h) = fullMul(x, y);
    uint256 mm = mulmod(x, y, d);
    if (mm > l) h -= 1;
    l -= mm;
    require(h < d, 'FullMath::mulDiv: overflow');
    return fullDiv(l, h, d);
}
```

Recommendation:

We advise the optimization to be applied to reduce the gas cost involved with the library.

Alleviation:

The optimization of Uniswap V2 was properly integrated into the codebase.

View Fix on GitHub

PREV
ERC20Permit.sol (ERP-C)

NEXT
Governable.sol (GOV-C)

Governable Code Style Findings

ON THIS PAGE

GOV-01C: Inexistent Deletion of Pending Governor

Type	Severity	Location
Gas Optimization	Informational	Governable.sol:L43-L47

Description:

The `_newGovernor` entry should be deleted when it is consumed by the `pullGovernor` function.

Example:

contracts/types/Governable.sol

SOL

Copy

```sol
43  function pullGovernor() public virtual override {
44      require( msg.sender == _newGovernor, "Governable: must be new governor to
45      emit GovernorPulled( _governor, _newGovernor );
46      _governor = _newGovernor;
47  }
```

Recommendation:

We advise it to be deleted so to ensure a consistent contract state.

Alleviation:

The new governor is now properly deleted when the `pullGovernor` function concludes.

View Fix on GitHub

PREV
FullMath.sol (FMH-C)

NEXT
Guardable.sol (GUA-C)

Guardable Code Style Findings

ON THIS PAGE

GUA-01C: Inexistent Deletion of Pending Guardian

Type	Severity	Location
Gas Optimization	Informational	Guardable.sol:L42-L46

Description:

The `_newGuardian` entry should be deleted when it is consumed by the `pullGuardian` function.

Example:

contracts/types/Guardable.sol

SOL

Copy

```sol
function pullGuardian() public virtual override {
    require( msg.sender == _newGuardian, "Guardable: must be new guardian to
    emit GuardianPulled( _guardian, _newGuardian );
    _guardian = _newGuardian;
}
```

Recommendation:

We advise it to be deleted so to ensure a consistent contract state.

Alleviation:

The new guardian is now properly deleted when the `pullGuardian` function concludes.

View Fix on GitHub

PREV
Governable.sol (GOV-C)

NEXT
ManagerOwnable.sol (MOE-C)

ManagerOwnable Code Style Findings

ON THIS PAGE

MOE-01C: Redundant Implementation

Type	Severity	Location
Code Style	Informational	ManagerOwnable.sol:L7-L10

Description:

The `ManagerOwnable` contract is redundant as it declares a new `modifier` labelled `onlyManager` that is exactly the same as the `onlyOwner` modifier, inclusive of the error messages.

Example:

contracts/types/ManagerOwnable.sol

SOL

Copy

```sol
import "./Ownable.sol";

contract ManagerOwnable is Ownable {
    modifier onlyManager() {
        require( _owner == msg.sender, "Ownable: caller is not the owner" );
        _;
    }
}
```

Recommendation:

We advise the implementation to be omitted from the codebase entirely.

Alleviation:

The contract is no longer part of the codebase rendering this exhibit null.

View Fix on GitHub

PREV
Guardable.sol (GUA-C)

NEXT
OlympusERC20.sol (OER-C)

OlympusERC20 Code Style Findings

ON THIS PAGE

OER-01C: Incorrect Function Visibility

Type	Severity	Location
Code Style	Informational	OlympusERC20.sol:L34

Description:

The `_burnFrom` function is incorrectly available externally by its `public` modifier.

Example:

contracts/OlympusERC20.sol

SOL

Copy

```sol
30  function burnFrom(address account_, uint256 amount_) public virtual {
31      _burnFrom(account_, amount_);
32  }
33
34  function _burnFrom(address account_, uint256 amount_) public virtual {
35      uint256 decreasedAllowance_ =
36          allowance(account_, msg.sender).sub(
37              amount_,
38              "ERC20: burn amount exceeds allowance"
39          );
40
41      _approve(account_, msg.sender, decreasedAllowance_);
42      _burn(account_, amount_);
43  }
```

Recommendation:

We advise it to be set to `internal` to properly illustrate its purpose and avoid potential circumventions of the `burnFrom` function in the future.

Alleviation:

The visibility specifier of the `_burnFrom` function was adjusted according to our recommendation.

View Fix on GitHub

PREV
ManagerOwnable.sol (MOE-C)

NEXT
OlympusTokenMigrator.sol (OTM-C)

OlympusTokenMigrator Code Style Findings

ON THIS PAGE

OTM-01C: Inexistent Error Messages

Type	Severity	Location
Code Style	Informational	OlympusTokenMigrator.sol:L99, L101, L103, L105, L107, L109, L111, L210, L238, L239, L328, L330, L332

Description:

The linked `require` checks have no explicit error messages defined.

Example:

contracts/migration/OlympusTokenMigrator.sol

SOL

Copy

```sol
208 // withdraw backing of migrated OHM
209 function defund(address reserve) external onlyGovernor {
210     require(ohmMigrated && timelockEnd < block.number && timelockEnd != 0);
211     oldwsOHM.unwrap(oldwsOHM.balanceOf(address(this)));
212
213     uint256 amountToUnstake = oldsOHM.balanceOf(address(this));
214     oldsOHM.approve(address(oldStaking), amountToUnstake);
215     oldStaking.unstake(amountToUnstake, false);
216
```

```
217     uint256 balance = oldOHM.balanceOf(address(this));
218
219     oldSupply = oldSupply.sub(balance);
220
221     uint256 amountToWithdraw = balance.mul(1e9);
222     oldOHM.approve(address(oldTreasury), amountToWithdraw);
223     oldTreasury.withdraw(amountToWithdraw, reserve);
224     IERC20(reserve).safeTransfer(address(newTreasury), IERC20(reserve).balanc
225
226     emit Defunded(balance);
227 }
```

Recommendation:

We advise them to be set so to aid in the validation of the conditionals as well as in debugging the application.

Alleviation:

Error messages were included to all `require` checks across the contract.

View Fix on GitHub

OTM-02C: Multiple Top-Level Declarations

Type	Severity	Location
Code Style	Informational	OlympusTokenMigrator.sol:L17-L44, L46-L50

Description:

The linked `interface` implementations should be relocated to dedicated files to not pollute the top-level of the contract file.

Example:

contracts/migration/OlympusTokenMigrator.sol

SOL Copy

```sol
interface IUniswapV2Router {
    function addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
        external
        returns (
            uint256 amountA,
            uint256 amountB,
            uint256 liquidity
        );

    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
```

```
41          address to,
42          uint256 deadline
43      ) external returns (uint256 amountA, uint256 amountB);
44  }
45
46  interface IStakingV1 {
47      function unstake(uint256 _amount, bool _trigger) external;
48
49      function index() external view returns (uint256);
50  }
51
52  contract OlympusTokenMigrator is OlympusAccessControlled {
```

Recommendation:

We advise them to be relocated to the `interfaces` sub-folder, potentially under an `external` second-level subfolder, to ensure that the code structure of the system is maintainable.

Alleviation:

All required interfaces were split to their dedicated files and are now properly imported to the codebase.

View Fix on GitHub

OTM-03C: Redundant & Confusing Comparisons

Type	Severity	Location
Gas Optimization	Informational ●	OlympusTokenMigrator.sol:L137, L182

Description:

The `if-else` structure within `migrate` and `_send` evaluate all states of the `enum` redundantly which can also cause ambiguous behaviour if the compiler does not enforce the value range of the `TYPE` enum due to a compiler issue given that it is internally represented by a `uint8`.

Example:

contracts/migration/OlympusTokenMigrator.sol

SOL Copy

```sol
118 enum TYPE {
119     UNSTAKED,
120     STAKED,
121     WRAPPED
122 }
123
124 // migrate OHMv1, sOHMv1, or wsOHM for OHMv2, sOHMv2, or gOHM
125 function migrate(
126     uint256 _amount,
127     TYPE _from,
128     TYPE _to
129 ) external {
130     uint256 sAmount = _amount;
131     uint256 wAmount = oldwsOHM.sOHMTowOHM(_amount);
132
133     if (_from == TYPE.UNSTAKED) {
134         oldOHM.safeTransferFrom(msg.sender, address(this), _amount);
135     } else if (_from == TYPE.STAKED) {
136         oldsOHM.safeTransferFrom(msg.sender, address(this), _amount);
137     } else if (_from == TYPE.WRAPPED) {
138         oldwsOHM.safeTransferFrom(msg.sender, address(this), _amount);
139         wAmount = _amount;
```

```
140             sAmount = oldwsOHM.wOHMToSOHM(_amount);
141         }
142
143         if (ohmMigrated) {
144             require(oldSupply >= oldOHM.totalSupply(), "OHMv1 minted");
145             _send(wAmount, _to);
146         } else {
147             gOHM.mint(msg.sender, wAmount);
148         }
149     }
```

Recommendation:

We advise the last `else if` branch to be converted to an `else` branch to ensure transfer of funds is performed at all times from the user to the contract and vice versa for the migration to occur.

Alleviation:

The `if-else` optimization was only applied to the first linked segment thereby partially alleviating this exhibit.

PREV
OlympusERC20.sol (OER-C)

NEXT
Ownable.sol (OWN-C)

Ownable Code Style Findings

ON THIS PAGE

OWN-01C: Inexistent Deletion of Pending Owner

Type	Severity	Location
Gas Optimization	Informational	Ownable.sol:L39-L43

Description:

The `_newOwner` entry should be deleted when it is consumed by the `pullManagement` function.

Example:

contracts/types/Ownable.sol

SOL

Copy

```sol
function pullManagement() public virtual override {
    require( msg.sender == _newOwner, "Ownable: must be new owner to pull");
    emit OwnershipPulled( _owner, _newOwner );
    _owner = _newOwner;
}
```

Recommendation:

We advise it to be deleted so to ensure a consistent contract state.

Alleviation:

The new owner is now properly deleted when the `pullManagement` function concludes.

View Fix on GitHub

PREV
OlympusTokenMigrator.sol (OTM-C)

NEXT
SafeMath.sol (SMH-C)

SafeMath Code Style Findings

ON THIS PAGE

SMH-01C: Inefficient Implementation

Type	Severity	Location
Gas Optimization	Informational	SafeMath.sol:L115-L127

Description:

The `sqqrt` implementation does not efficiently calculate the root of the provided argument as it wraps operations unnecessarily (i.e. divisions with non-zero value literals).

Example:

contracts/libraries/SafeMath.sol

SOL

Copy

```sol
// Only used in the  BondingCalculator.sol
function sqrrt(uint256 a) internal pure returns (uint c) {
    if (a > 3) {
        c = a;
        uint b = add( div( a, 2), 1 );
        while (b < c) {
            c = b;
            b = div( add( div( a, b ), b), 2 );
        }
    } else if (a != 0) {
        c = 1;
    }
}
```

Recommendation:

We advise a more efficient square root algorithm to be implemented instead, such as the optimized Babylonian method by Uniswap.

Alleviation:

The Olympus DAO team considered this exhibit but opted not to apply any remediation for it.

PREV
Ownable.sol (OWN-C)

NEXT
StakingDistributor.sol (SDR-C)

StakingDistributor Code Style Findings

ON THIS PAGE

SDR-01C: Inexistent Error Messages

Type	Severity	Location
Code Style	Informational	StakingDistributor.sol:L48, L50, L52, L133, L144

Description:

The linked `require` checks contain no descriptive error messages.

Example:

contracts/StakingDistributor.sol

SOL

Copy

```sol
constructor(
    address _treasury,
    address _ohm,
    address _staking
) {
    require(_treasury != address(0));
    treasury = ITreasury(_treasury);
    require(_ohm != address(0));
    OHM = IERC20(_ohm);
    require(_staking != address(0));
    staking = _staking;
}
```

Recommendation:

We advise them to be set so to aid in the debugging of the application and to also enable more accurate validation of the `require` condition purposes.

Alleviation:

Error messages were introduced in all linked `require` checks.

View Fix on GitHub

SDR-02C: Inexistent Variable Visibility Specifiers

Type	Severity	Location
Code Style	Informational	StakingDistributor.sol:L21, L22, L23

Description:

The linked variables have no visibility specifier explicitly set.

Example:

contracts/StakingDistributor.sol

SOL

Copy

```sol
21  IERC20 immutable OHM;
22  ITreasury immutable treasury;
23  address immutable staking;
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current compiler behaviour is to assign a specifier automatically.

Alleviation:

Proper visibility specifiers were set for all linked variables.

View Fix on GitHub

PREV
SafeMath.sol (SMH-C)

NEXT
StandardBondingCalculator.sol (SBC-C)

StandardBondingCalculator Code Style Findings

ON THIS PAGE

SBC-01C: Inexistent Error Message

Type	Severity	Location
Code Style	Informational	StandardBondingCalculator.sol:L26

Description:

The linked `require` check contains no descriptive error message.

Example:

contracts/StandardBondingCalculator.sol

SOL

Copy

```
26  require( _OHM != address(0) );
```

Recommendation:

We advise one to be set so to aid in the debugging of the application and to also enable more accurate validation of the `require` condition's purpose.

Alleviation:

Error messages were introduced in all linked `require` checks.

View Fix on GitHub

SBC-02C: Inexistent Variable Visibility Specifier

Type	Severity	Location
Code Style	Informational	StandardBondingCalculator.sol:L23

Description:

The linked variable has no visibility specifier explicitly set.

Example:

contracts/StandardBondingCalculator.sol

SOL Copy

```
23  IERC20 immutable OHM;
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current compiler behaviour is to assign a specifier automatically.

Alleviation:

The `internal` visibility specifier was properly introduced to the linked variable.

View Fix on GitHub

PREV
StakingDistributor.sol (SDR-C)

NEXT
Timelock.sol (TIM-C)

Timelock Code Style Findings

ON THIS PAGE

TIM-01C: Redundant Implementation

Type	Severity	Location
Language Specific	Informational	Timelock.sol:L53-L64

Description:

The `sqrrt` implementation present within the in-file declared `SafeMath` function is redundant and overly convoluted by wrapping each statement with its `SafeMath` equivalent, which at times is unnecessary such as when dividing with non-zero value literals.

Example:

contracts/governance/Timelock.sol

SOL

Copy

```sol
function sqrrt(uint256 a) internal pure returns (uint c) {
    if (a > 3) {
        c = a;
        uint b = add( div( a, 2), 1 );
        while (b < c) {
            c = b;
            b = div( add( div( a, b ), b), 2 );
        }
    } else if (a != 0) {
        c = 1;
    }
}
```

Recommendation:

We advise the implementation to be entirely omitted to also ensure that source code match analysis detects the `Timelock` contract as being an identical copy of Compound's implementation.

Alleviation:

The `sqrrt` function was safely omitted from the codebase.

View Fix on GitHub

PREV
StandardBondingCalculator.sol (SBC-C)

NEXT
Treasury.sol (TRE-C)

Treasury Code Style Findings

ON THIS PAGE

TRE-01C: Improper Failure Enforcement

Type	Severity	Location
Code Style	Informational ●	Treasury.sol:L106

Description:

The `require` check linked performs a guaranteed-to-fail check (`1 == 0`) to illustrate the error message that accompanies it.

Example:

contracts/Treasury.sol

SOL

Copy

```sol
106 require(1 == 0, "neither reserve nor liquidity token"); // guarantee revert
```

Recommendation:

We advise a `revert` to be used directly instead that accepts the error message directly.

Alleviation:

The `require` check was substituted for a `revert` check as recommended

The `require` check was substituted for a `revert` check as recommended.

View Fix on GitHub

TRE-02C: Improper Permitted Execution Flow

Type	Severity	Location
Code Style	Informational	Treasury.sol:L353-L362

Description:

The `enableOnChainGovernance` function should not be invoke-able if the `onChainGoverned` status has already been set.

Example:

contracts/Treasury.sol

SOL

Copy

```sol
353 /**
354  * @notice disables timelocked functions
355  */
356 function enableOnChainGovernance() external onlyOwner {
357     if (onChainGovernanceTimelock != 0 && onChainGovernanceTimelock <= block.
358         onChainGoverned = true;
359     } else {
360         onChainGovernanceTimelock = block.number.add(blocksNeededForQueue.mul
361     }
362 }
```

Recommendation:

We advise this to be enforced by introducing a `require` check that prevents this scenario at the top of the function.

Alleviation:

The function now properly validates that `onChainGoverned` has not been set already.

View Fix on GitHub

TRE-03C: Inexistent Error Messages

Type	Severity	Location
Code Style	Informational	Treasury.sol:L81, L150, L305, L321

Description:

The linked `require` checks contain no descriptive error messages.

Example:

contracts/Treasury.sol

SOL

Copy

```sol
constructor(address _OHM, uint256 _timelock) {
    require(_OHM != address(0));
    OHM = IOHMERC20(_OHM);

    blocksNeededForQueue = _timelock;
}
```

Recommendation:

We advise them to be set so to aid in the debugging of the application and to also enable more accurate validation of the `require` condition purposes.

Alleviation:

Error messages were introduced in all linked `require` checks.

View Fix on GitHub

TRE-04C: Inexistent Variable Visibility Specifier

Type	Severity	Location
Code Style	Informational	Treasury.sol:L60

Description:

The linked variable has no visibility specifier explicitly set.

Example:

contracts/Treasury.sol

SOL Copy

```sol
60  IOHMERC20 immutable OHM;
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current compiler behaviour is to assign a specifier automatically.

Alleviation:

The `public` visibility specifier was explicitly set to the linked variable.

View Fix on GitHub

PREV
Timelock.sol (TIM-C)

NEXT
gOHM.sol (OHM-C)

gOHM Code Style Findings

ON THIS PAGE

OHM-01C: Inexistent Error Messages

Type	Severity	Location
Code Style	Informational	gOHM.sol:L59, L127, L129, L132

Description:

The linked `require` checks contain no descriptive error messages.

Example:

contracts/governance/gOHM.sol

SOL

Copy

```sol
58  constructor(address _migrator) {
59      require(_migrator != address(0));
60      approved = _migrator;
61  }
```

Recommendation:

We advise them to be set so to aid in the debugging of the application and to also enable more accurate validation of the `require` condition purposes.

Alleviation:

Error messages were introduced in all linked `require` checks.

View Fix on GitHub

PREV
Treasury.sol (TRE-C)

NEXT
sOlympusERC20.sol (OEC-C)

sOlympusERC20 Code Style Findings

ON THIS PAGE

OEC-01C: Deprecated Representation Style

Type	Severity	Location
Code Style	Informational ●	sOlympusERC20.sol:L54

Description:

The maximum of `uint256` representation in use (`~uint256(0)`) has been deprecated in favor of the special `type` operator.

Example:

contracts/sOlympusERC20.sol

SOL Copy

```
54  uint256 private constant MAX_UINT256 = ~uint256(0);
```

Recommendation:

We advise the operator to be used to instead assign the maximum (`type(uint256).max`).

Alleviation:

The representation of the maximum was adjusted according to our recommendation.

View Fix on GitHub

OEC-02C: Inefficient Code Structure

Type	Severity	Location
Gas Optimization	Informational	sOlympusERC20.sol:L174, L175, L185, L186, L191, L192, L198-L203, L210-L213

Description:

All `approve` style functions can internally use the `_approve` function.

Example:

contracts/sOlympusERC20.sol

SOL

Copy

```sol
184 function approve( address spender, uint256 value ) public override returns (b
185     _allowedValue[ msg.sender ][ spender ] = value;
186     emit Approval( msg.sender, spender, value );
187     return true;
188 }
189
190 function increaseAllowance( address spender, uint256 addedValue ) public over
191     _allowedValue[ msg.sender ][ spender ] = _allowedValue[ msg.sender ][ spe
192     emit Approval( msg.sender, spender, _allowedValue[ msg.sender ][ spender
193     return true;
194 }
195
196 function decreaseAllowance( address spender, uint256 subtractedValue ) public
197     uint256 oldValue = _allowedValue[ msg.sender ][ spender ];
198     if (subtractedValue >= oldValue) {
199         _allowedValue[ msg.sender ][ spender ] = 0;
200     } else {
201         _allowedValue[ msg.sender ][ spender ] = oldValue.sub( subtractedValu
202     }
203     emit Approval( msg.sender, spender, _allowedValue[ msg.sender ][ spender
204     return true;
205 }
206
207 /* ========== INTERNAL FUNCTIONS ========== */
```

```
208
209 // called in a permit
210 function _approve( address owner, address spender, uint256 value ) internal o
211     _allowedValue[owner][spender] = value;
212     emit Approval( owner, spender, value );
213 }
```

Recommendation:

We advise them to do so to signficantly reduce the bytecode size of the contract.

Alleviation:

The code was refactored to properly utilize `_approve` in all instances possible.

View Fix on GitHub

OEC-03C: Inexistent Error Messages

Type	Severity	Location
Code Style	Informational	sOlympusERC20.sol:L80, L81, L86, L87, L88, L94, L96

Description:

The linked `require` checks contain no descriptive error messages.

Example:

contracts/sOlympusERC20.sol

SOL

Copy

```sol
79  function setIndex( uint _INDEX ) external {
80      require( msg.sender == initializer );
81      require( INDEX == 0 );
82      INDEX = gonsForBalance( _INDEX );
83  }
84
85  function setgOHM( address _gOHM ) external {
86      require( msg.sender == initializer );
87      require( address( gOHM ) == address(0) );
88      require( _gOHM != address(0) );
89      gOHM = IgOHM( _gOHM );
90  }
91
92  // do this last
93  function initialize( address stakingContract_ ) external {
94      require( msg.sender == initializer );
95
96      require( stakingContract_ != address(0) );
97      stakingContract = stakingContract_;
98      _gonBalances[ stakingContract ] = TOTAL_GONS;
99
100     emit Transfer( address(0x0), stakingContract, _totalSupply );
101     emit LogStakingContractUpdated( stakingContract_ );
102
103     initializer = address(0);
```

```
104 }
```

Recommendation:

We advise them to be set so to aid in the debugging of the application and to also enable more accurate validation of the `require` conditions.

Alleviation:

Error messages were introduced in all linked `require` checks.

View Fix on GitHub

OEC-04C: Inexistent Variable Visibility Specifiers

Type	Severity	Location
Code Style	Informational ●	sOlympusERC20.sol:L45, L47

Description:

The linked variables have no visibility specifier explicitly set.

Example:

contracts/sOlympusERC20.sol

SOL Copy

```sol
45  address initializer;
46
47  uint INDEX; // Index Gons - tracks rebase growth
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current compiler behaviour is to assign a specifier automatically.

Alleviation:

The Olympus DAO team considered this exhibit but opted not to apply any remediation for it.

OEC-05C: Redundant Event Argument

Type	Severity	Location
Code Style	Informational ●	sOlympusERC20.sol:L20, L118, L159

Description:

The current `timestamp` that a `LogSupply` event emits is already attached to each event emittance by the blockchain itself.

Example:

contracts/sOlympusERC20.sol

SOL Copy

```sol
20  event LogSupply(uint256 indexed epoch, uint256 timestamp, uint256 totalSupply
```

Recommendation:

We advise the `timestamp` member to be omitted from the event as it is redundant.

Alleviation:

The redundant event argument was safely omitted from the codebase.

View Fix on GitHub

PREV
gOHM.sol (OHM-C)

NEXT
Finding Types

ID	Severity	Addressed	Title
GUA-01C	Informational	Yes	Inexistent Deletion of Pending Guardian
MOE-01C	Informational	Yes	Redundant Implementation
OER-01C	Informational	Yes	Incorrect Function Visibility
OTM-01C	Informational	Yes	Inexistent Error Messages
OTM-02C	Informational	Yes	Multiple Top-Level Declarations
OTM-03C	Informational	No	Redundant & Confusing Comparisons
OWN-01C	Informational	Yes	Inexistent Deletion of Pending Owner
SMH-01C	Informational	No	Inefficient Implementation
SDR-01C	Informational	Yes	Inexistent Error Messages
SDR-02C	Informational	Yes	Inexistent Variable Visibility Specifiers
SBC-01C	Informational	Yes	Inexistent Error Message
SBC-02C	Informational	Yes	Inexistent Variable Visibility Specifier
TIM-01C	Informational	Yes	Redundant Implementation
TRE-01C	Informational	Yes	Improper Failure Enforcement
TRE-02C	Informational	Yes	Improper Permitted Execution Flow
TRE-03C	Informational	Yes	Inexistent Error Messages
TRE-04C	Informational	Yes	Inexistent Variable Visibility Specifier
OHM-01C	Informational	Yes	Inexistent Error Messages
OEC-01C	Informational	Yes	Deprecated Representation Style

ID	Severity	Addressed	Title
OEC-02C	Informational	Yes	Inefficient Code Structure
OEC-03C	Informational	Yes	Inexistent Error Messages
OEC-04C	Informational	No	Inexistent Variable Visibility Specifiers
OEC-05C	Informational	Yes	Redundant Event Argument

PREV
Manual Review

NEXT
BondTeller.sol (BTR-S)

Finding Types

ON THIS PAGE

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

External Call Validation

Many contracts that interact with DeFi contain a set of complex external call executions that need to happen in a particular sequence and whose execution is usually taken for granted whereby it is not always the case. External calls should always be validated, either in the form of `require` checks imposed at the contract-level or via more intricate mechanisms such as invoking an external getter-variable and ensuring that it has been properly updated.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted `if` blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Solidity language boasts that discerns it from other conventional programming languages. For example, the EVM is a 256-bit machine meaning that operations on less-than-256-bit types are more costly for the EVM in terms of gas costs, meaning that loops utilizing a `uint8` variable because their limit will never exceed the 8-bit range actually cost more than redundantly using a `uint256` variable.

Code Style

An official Solidity style guide exists that is constantly under development and is adjusted on each new Solidity release, designating how the overall look and feel of a codebase should be. In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a contract-level variable that is present in the inheritance chain of the local execution level's context.

Gas Optimization

Gas optimization findings relate to ways the codebase can be optimized to reduce the gas cost involved with interacting with it to various degrees. These types of findings are completely optional and are pointed out for the benefit of the project's developers.

Standard Conformity

These types of findings relate to incompatibility between a particular standard's implementation and the project's implementation, oftentimes causing significant issues in the usability of the contracts.

Mathematical Operations

In Solidity, math generally behaves differently than other programming languages due to the constraints of the EVM. A prime example of this difference is the truncation of values during a division which in turn leads to loss of precision and can cause systems to behave incorrectly when dealing with percentages and proportion calculations.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

PREV
sOlympusERC20.sol (OEC-C)

NEXT
Source Code