4rena

Olympus DAO contest
Findings & Analysis Report

2022-10-26

TABLE OF CONTENTS

Top

Overview

ABOUT C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the Olympus DAO smart contract system written in Solidity. The audit contest took place between August 25—September 1 2022.

WARDENS

156 Wardens contributed reports to the Olympus DAO contest:

1. zzzitron
2. 0x52
3. Trust
4. rbserver
5. Lambda
6. enckrish
7. 0x1f8b
8. IllIllI
9. reassor
10. cryptphi
11. datapunk
12. rvierdiiev
13. Bahurum
14. minhtrng
15. immeas
16. Czar102
17. bin2chen
18. csanuragjain
19. cccz
20. hansfriese
21. Jeiwan
22. berndartmueller

23. itsmeSTYJ
24. brgltd
25. d3e4
26. djxploit
27. V_B (Barichek and vlad_bochok)
28. GalloDaSballo
29. m9800
30. Aymen0909
31. hyh
32. ladboy233
33. carlitox477
34. 0xNazgul
35. Ruhum
36. sorrynotsorry
37. pedroais
38. pashov
39. __141345__
40. CertoraInc (egjlmn1, OriDabush, ItayG, shakedwinder, and RoiEvenHaim)
41. tonisives
42. 0xSky
43. PwnPatrol (obront and throttle)
44. okkothejawa
45. pfapostol
46. c3phas
47. yixxas
48. 0xSmartContract
49. Guardian
50. devtooligan
51. 0xNineDec
52. LeoS
53. Tomo
54. Deivitto
55. ReyAdmirado

56. TomJ
57. Sm4rty
58. gogo
59. Rolezn
60. ignacio
61. ret2basic
62. oyc_109
63. ajtra
64. 0xDjango
65. Bnke0x0
66. grGred
67. robee
68. 0xkatana
69. fatherOfBlocks
70. erictee
71. 0x040
72. ElKu
73. cRat1st0s
74. durianSausage
75. lukris02
76. martin
77. Rohan16
78. sikorico
79. tnevler
80. StevenL
81. RaymondFam
82. Waze
83. delfin454000
84. medikko
85. bobirichman
86. CodingNameKiki
87. Chandr
88. rokinot

89. 0x85102
90. aviggiano
91. apostle0x01
92. Funen
93. natzuu
94. The_GUILD (David_, Ephraim, LeoGold, and greatsamist)
95. JansenC
96. Diraco
97. ne0n
98. mics
99. ak1
100. shenwilly
101. m_Rassska
102. dipp
103. DimSon
104. nxrblsrpr
105. BipinSah
106. Ch_301
107. prasantgupta52
108. w0Lfrum
109. rajatbeladiya
110. ch13fd357r0y3r
111. PPrieditis
112. Chom
113. eierina
114. PaludoX0
115. Picodes
116. p_crypt0
117. Margaret
118. 8olidity
119. EthLedger
120. indijanc
121. CRYP70

122. cloudjunky
123. MasterCookie
124. JC
125. exolorkistis
126. zishansami
127. Dionysus
128. ch0bu
129. jag
130. Noah3o6
131. Saintcode_
132. chrisdior4
133. Amithuddar
134. Shishigami
135. Metatron
136. RoiEvenHaim
137. peiw
138. karanctf
139. kris
140. simon135
141. Tagir2003
142. SooYa
143. newfork01
144. Fitraldys
145. Dravee
146. Jujic
147. peachtea

This contest was judged by 0xean.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 35 unique vulnerabilities. Of these vulnerabilities, 3 received a risk rating in the category of HIGH severity and 32 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 114 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 91 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Olympus DAO contest repository, and is composed of 18 smart contracts written in the Solidity programming language and includes 1,944 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (3)

[H-01] IN `GOVERNANCE.SOL`, IT MIGHT BE IMPOSSIBLE TO ACTIVATE A NEW PROPOSAL FOREVER AFTER FAILED TO EXECUTE THE

PREVIOUS ACTIVE PROPOSAL.

Submitted by hansfriese, also found by berndartmueller, csanuragjain, m9800, V_B, and zzzitron

https://github.com/code-423n4/2022-08-olympus/blob/2a0b515012b4a40076f6eac487f7816aafb8724a/src/policies/Governance.sol#L216-L221
https://github.com/code-423n4/2022-08-olympus/blob/2a0b515012b4a40076f6eac487f7816aafb8724a/src/policies/Governance.sol#L302-L304

Currently, if users vote for the active proposal, the `VOTES` are transferred to the contract so that users can't vote or endorse other proposals while the voted proposal is active.

And the active proposal can be replaced only when the proposal is executed successfully or another proposal is activated after `GRACE_PERIOD`.

But `activateProposal()` requires at least 20% endorsements here, so it might be impossible to activate a new proposal forever if the current active proposal involves more than 80% of total votes.

Proof of Concept

The below scenario would be possible.

1. `Proposal 1` was submitted and activated successfully.
2. Let's assume 81% of the total votes voted for this proposal. `Yes = 47%`, `No = 34%`
3. This proposal can't be executed for this requirement because `47% - 34% = 13% < 33%`.
4. Currently the contract contains more than 81% of total votes and users have at most 19% in total.
5. Also users can't reclaim their votes among 81% while `Proposal 1` is active.
6. So even if a user who has 1% votes submits a new proposal, it's impossible to activate because of this require().
7. So it's impossible to delete `Proposal 1` from an active proposal and there won't be other active proposal forever.

Tools Used

Solidity Visual Developer of VSCode

Recommended Mitigation Steps

I think we should add one more constant like `EXECUTION_EXPIRE = 2 weeks` so that voters can reclaim their votes after this period even if the proposal is active.

I am not sure we can use the current `GRACE_PERIOD` for that purpose.

So `reclaimVotes()` should be modified like below.

```
    function reclaimVotes(uint256 proposalId_) external {
        uint256 userVotes = userVotesForProposal[proposalId_][msg.sender];

        if (userVotes == 0) {
            revert CannotReclaimZeroVotes();
        }

        if (proposalId_ == activeProposal.proposalId) {
            if (block.timestamp < activeProposal.activationTimestamp + EXECUTION_EXPIRE) //
            {
                revert CannotReclaimTokensForActiveVote();
            }
        }

        if (tokenClaimsForProposal[proposalId_][msg.sender] == true) {
            revert VotingTokensAlreadyReclaimed();
        }

        tokenClaimsForProposal[proposalId_][msg.sender] = true;

        VOTES.transferFrom(address(this), msg.sender, userVotes);
    }
```

fullyallocated (Olympus) confirmed

[H-02] ANYONE CAN PASS ANY PROPOSAL ALONE BEFORE FIRST `VOTES` ARE MINTED

Submitted by Bahurum, also found by bin2chen and cryptphi

https://github.com/code-423n4/2022-08-olympus/blob/2a0b515012b4a40076f6eac487f7816aafb8724a/src/policies/Governance.sol#L164
https://github.com/code-423n4/2022-08-olympus/blob/2a0b515012b4a40076f6eac487f7816aafb8724a/src/policies/Governance.sol#L217-L218
https://github.com/code-423n4/2022-08-olympus/blob/2a0b515012b4a40076f6eac487f7816aafb8724a/src/policies/Governance.sol#L268

Before any `VOTES` are minted anyone can activate and execute an arbitrary proposal even with 0 votes cast. So an attacker can pass any proposal (i.e. change the `executor` + `admin` of the `Kernel`, gaining access to all permissioned functions and to funds held).

Proof of Concept

Checks on vote numbers made in `Governance.sol` at lines L164, 217-218, 268 pass if `VOTES.totalSupply() == 0`. So, until no `VOTES` are minted, anyone can submit, activate and execute a proposal. There is no need to own or cast votes. This happens if `OlympusGovernance` is granted the `executor` role before any `VOTES` are minted (as in Governance.t.sol). The attacker can anticipate/frontrun the minting and pass a proposal to change both the `Kernel` `admin` and `executor`. Then he/she can upgrade malicious modules, steal funds from treasury...

A PoC was obtained modifying the `setUp()` of Governance.t.sol by keeping only what is before the minting of `VOTES` (up to L83 included). The test is as follows:

```
function test_AttackerPassesProposalBeforeMinting() public {

    address[] memory users = userCreator.create(1);
    address attacker = users[0];
    vm.prank(attacker);
    MockMalicious attackerControlledContract = new MockMalicious();

    Instruction[] memory instructions_ = new Instruction[](2);
    instructions_[0] = Instruction(Actions.ChangeAdmin, address(attackerControlledCo
    instructions_[1] = Instruction(Actions.ChangeExecutor, address(attackerControlle

    vm.prank(attacker);
    governance.submitProposal(instructions_, "proposalName", "This is the proposal U

    governance.endorseProposal(1);

    vm.prank(attacker);
    governance.activateProposal(1);

    vm.warp(block.timestamp + 3 days + 1);

    governance.executeProposal();

    assert(kernel.executor()==address(attackerControlledContract));
    assert(kernel.admin()==address(attackerControlledContract));


}
```

with

```
contract MockMalicious {}
```

Recommended Mitigation Steps

In `Governance.sol` check for a minimum VOTES totalSupply, similiar to the expected initial supply of VOTES when they have been fairly distributed, for example at line L164.

fullyallocated (Olympus) acknowledged

0xean (judge) commented:

> Leaving as High severity as this shows a clear path to loss of funds.

[H-03] TRSRY: FRONT-RUNNABLE SETAPPROVALFOR

Submitted by zzzitron, also found by berndartmueller, csanuragjain, pashov, Ruhum, sorrynotsorry, and Trust

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/modules/TRSRY.sol#L64-L72
https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/TreasuryCustodian.sol#L4
L48

An attacker may be able to withdraw more than intended

Proof of Concept

Let’s say Alice had approval of 100. Now the treasury custodian reduced the approval to 50. Alice could frontrun the `setApprovalFor` of 50, and withdraw 100 as it was before. Then withdraw 50 with the newly set approval. So the alice could withdraw 150.

```
// modules/TRSRY.sol

 63     /// @notice Sets approval for specific withdrawer addresses
 64     function setApprovalFor(
 65         address withdrawer_,
 66         ERC20 token_,
 67         uint256 amount_
 68     ) external permissioned {
 69         withdrawApproval[withdrawer_][token_] = amount_;
 70 
 71         emit ApprovedForWithdrawal(withdrawer_, token_, amount_);
 72     }
```

The `TreasuryCustodian` simply calls the `setApprovalFor` to grant Approval.

```
 41
```

```
    function grantApproval(
        address for_,
        ERC20 token_,
        uint256 amount_
    ) external onlyRole("custodian") {
        TRSRY.setApprovalFor(for_, token_, amount_);
    }
```

Recommended Mitigation Steps

Instead of setting the given amount, one can reduce from the current approval. By doing so, it checks whether the previous approval is spend.

ind-igo (Olympus) confirmed and commented:

> Understood. Will change the logic to increase/decrease allowances.

0xean (judge) increased severity to High and commented:

> I think this vulnerability should be a high severity as it opens up the possibility of a direct loss of funds in the amount of up to the previous approval amount. Upgrading to High.

0xean (judge) commented:

> @ind-igo - Not sure if you deleted your comment, but that context is useful. Happy to take another look here.

ind-igo (Olympus) commented:

> I did, I just thought it was unnecessary to evaluate the issue. I was just saying that the context of the code is that it is not intended to be used to approve an EOA/multisig, but instead used to approve governance-voted contracts to access treasury funds, in order to deposit into yield contracts or whatever. But I don't think it's very relevant to this, as the code is still faulty and exploitable in an extreme case. I already have made this remediation as well, so all good.

Medium Risk Findings (32)

[M-01] OPERATOR::SETRESERVEFACTOR DOESN'T CHECK IF BOND MARKET SHOULD BE CHANGED

Submitted by rvierdiiev

`Operator::setReserveFactor` sets new `reserveFactor` value. This parameter is used in `fullCapacity` function to calculate how much capacity is available by high/low side. Then calculated capacity is used by `Range` module inside `regenerate` function to set the threshold of capacity for both sides of market. Then in `Range::updateCapacity` function this threshold is checked to understand if the wall should be down and the bond market should be closed.

Changing this value means that the capacity of sides has changed and the sides should be regenarated to include this changes.

Proof of Concept

https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Operator.sol#L548
https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Operator.sol#L711
https://github.com/code-423n4/2022-08-olympus/blob/main/src/modules/RANGE.sol#L133
https://github.com/code-423n4/2022-08-olympus/blob/main/src/modules/RANGE.sol#L145
https://github.com/code-423n4/2022-08-olympus/blob/main/src/modules/RANGE.sol#L185
https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Operator.sol#L780

Recommended Mitigation Steps

Call this after the param updating.
`_regenerate(true); _regenerate(false;`

Oighty (Olympus) confirmed and commented:

> Forcing a regeneration when the reserveFactor is updated could cause unintended regeneration if a wall is currently down. A better approach may be to conditionally regenerate each side if they are active.

Oighty (Olympus) acknowledged and commented:

> After discussing with the team more, we are going to leave this as-is. It is more flexible to not regenerate the side in this function. With the current implementation, the guardian can determine if the change should go into effect on the next regen, or if it should happen immediately. To enable immediately, they can manually call `regenerate`.

[M-02] SOLMATE SAFETRANSFER AND SAFETRANSFERFROM DOES NOT CHECK THE CODESIZE OF THE TOKEN ADDRESS, WHICH MAY LEAD TO FUND LOSS

Submitted by djxploit, also found by brgltd

In `getloan()` and `replayloan()`, the `safetransfer` and `safetransferfrom` doesn't check the existence of code at the token address. This is a known issue while using solmate's libraries.

Hence this may lead to miscalculation of funds and may lead to loss of funds , because if `safetransfer()` and `safetransferfrom()` are called on a token address that doesn't have contract in it, it will always return success, bypassing the return value check. Due to this protocol will think that funds has been transferred and successful , and records will be accordingly calculated, but in reality funds were never transferred.
So this will lead to miscalculation and possibly loss of funds

Proof of Concept

https://github.com/code-423n4/2022-08-olympus/blob/main/src/modules/TRSRY.sol#L110
https://github.com/code-423n4/2022-08-olympus/blob/main/src/modules/TRSRY.sol#L99

Recommended Mitigation Steps

Use openzeppelin's safeERC20 or implement a code existence check.

ind-igo (Olympus) confirmed and commented:

> Confirmed. Will implement this. Thank you.

[M-03] RBS MAY REDEPLOY FUNDS AUTOMATICALLY IF PRICE STAYS ABOVE OR BELOW WALL FOR LONGER THAN `_CONFIG.REGENWAIT`

Submitted by 0x52

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Operator.sol#L195-L268

Loss of treasury funds.

Proof of Concept

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/modules/RANGE.sol#L133-L139

```
    if (capacity_ < _range.high.threshold && _range.high.active) {
        // Set wall to inactive
        _range.high.active = false;
        _range.high.lastActive = uint48(block.timestamp);


        emit WallDown(true, block.timestamp, capacity_);
    }
```

_range.high.lastActive and _range.low.lastActive are only updated in RANGE.sol when _range.x.capacity < _range.x.threshold and the _range.x.active == true. After this is tripped, _range.x.active will be set to false, meaning that _range.x.lastActive will not be updated again until the wall is regenerated and capacity is restored.

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Operator.sol#L209-L214

```
        if (
            uint48(block.timestamp) >= RANGE.lastActive(true) + uint48(config_.regenWait) &&
            _status.high.count >= config_.regenThreshold
        ) {
            _regenerate(true);
        }
```

If 1) the price were to sustain outside of the range (high volatility for volatile asset, black swan for stable) for longer than config_.regenWait and 2) config_regenThreshold satisfies the following equation:

```
config_.regenThreshold <= _config.regenObserve - config_.regenWait / frequency
```

then *status.high.count could be greater than config*.regenThreshold. This would trigger more funds to be deployed even though the price never came back inside the wall price.

In this scenario the wall price would be far from the true price of the asset leading to loss of treasury funds as it buys/sell at prices well above/below market price.

Recommended Mitigation Steps

A check should be added to verify that the price is within the wall price before regenerating. Alternatively, config_.regenTheshold could be set to satisfy the following equation:

```
config_.regenThreshold > _config.regenObserve - config_.regenWait / frequency
```

This would eliminate the risk as _status.high.count >= config_.regenThreshold could never be true for a sustained period where current price is greater than the wall price.

Oighty (Olympus) disagreed with severity and commented:

> This is valid. Our intended parameterization of the system would not be subject to this vulnerability, but it would be an issue if the system was incorrectly parameterized. Because it is an edge case, I'm not sure it is a high risk bug though.
>
> Another potential fix is resetting the `count` to 0 and the `observations` array to `new bool[](regenObserve)` to clear out positive values from when a wall goes down. This could be done in the `_updateCapacity()` function by checking if the new capacity is under the threshold.

0xean (judge) decreased severity to Medium and commented:

> Going to downgrade to Medium as the external dependency is a configuration that is not planned to be used by the sponsor.

[M-04] `OLYMPUSGOVERNANCE#EXECUTEPROPOSAL`: REENTRANCY ATTACK VULNERABLE FUNCTION

Submitted by carlitox477, also found by cryptphi and ladboy233

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Governance.sol#L265
https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Governance.sol#L278-L288

Given that the activeProposal change is done before the for loop, if this function is call through one `kernel.executeAction(instruction,target)` we can call the same instructions (in the same order) again and again, which may or may not affect funds (depending on the instructions).

Proof of Concept

For instance, if we install a new module, and this module has a vulnerability (even intentional), the next steps can by trigger:

1. Call executeAction
2. This allow us to call kernel.executeAction in the for loop
3. executAction allow us to call _installModule
4. _installModule allow us to call newModule_.Init

5. By init we can call now executeProposal again (suppose that the init function interact with a previous vulnerable proxy contract to scam voters to vote in favour of this proposal as if it was a contract which is ok, and before calling executeProposal we change the implementation to allow this attack),

Recommended Mitigation Steps

Use nonReentrant modifier or move the line `activeProposal = ActivatedProposal(0, 0);` before the for loop.

fullyallocated (Olympus) confirmed and commented:

> I don't know if funds are going to be threatened, but this does allow for a re-entrancy. Warden is correct in resetting the active Proposal before the for loop based on the checks-effects-interactions code design pattern.

[M-05] PROPOSALS OVERWRITE

Submitted by 0x1f8b

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Governance.sol#L167
https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Governance.sol#L66

It is possible to overwrite proposals in certain circumstances. The method `Governance.submitProposal` doesn't check if the `proposalId` (stored in a different contract) exists already as a valid proposal in `getProposalMetadata`.

Proof of Concept

If the project update the kernel module "`INSTR`" and reconfigure proposals and call `INSTR.store(instructions_);`, the counter may return a `proposalId` of an existing proposal and overwrite an existing previous one.

This is due to the fact that the proposals are saved in a mapping of a contract that is not related to the one that returns the counters, and furthermore, they do not check that the record already exists.

```
        uint256 proposalId = INSTR.store(instructions_);
        getProposalMetadata[proposalId] = ProposalMetadata(
            title_,
            msg.sender,
            block.timestamp,
            proposalURI_
        );
```

Recommended Mitigation Steps

- Store the proposal metadata in the same `INSTR` contract or ensure that the proposal doesn't exist.

fullyallocated (Olympus) acknowledged, but disagreed with severity and commented:

> Agreed with the validity of the circumstance, but it is contingent on us upgrading the contract in an unexpected way. Is the same as saying "if you upgrade a contract incorrectly it can break the dependencies".

0xean (judge) decreased severity to Medium and commented:

> Going to downgrade to medium based on some external requirements needing to be in place to be realized.
>
> ```
> Assets not at direct risk, but the function of the protocol or its availability
> could be impacted, or leak value with a hypothetical attack path with stated
> assumptions, but external requirements.
> ```
>
> Function of the protocol could be impacted and there are external requirements.

[M-06] AFTER ENDORSING A PROPOSAL, USER CAN TRANSFER VOTES TO ANOTHER USER FOR ENDORSING THE SAME PROPOSAL AGAIN

Submitted by rbserver, also found by 0x1f8b, Bahurum, csanuragjain, and yixxas

https://github.com/code-423n4/2022-08-olympus/blob/main/src/modules/VOTES.sol#L9-L11
https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Governance.sol#L180-L201
https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Governance.sol#L205-L236

The following comment indicates that the `OlympusVotes` contract is a stub for `gOHM`. Checking the `gOHM` contract at https://etherscan.io/token/0x0ab87046fBb341D058F17CBC4c1133F25a20a52f#code, the `transfer` and `transferFrom` functions are available.

https://github.com/code-423n4/2022-08-olympus/blob/main/src/modules/VOTES.sol#L9-L11

```
/// @notice Votes module is the ERC20 token that represents voting power in the network.
```

```
/// @dev    This is currently a substitute module that stubs gOHM.
contract OlympusVotes is Module, ERC20 {
```

Moreover, the documentation states that the vote redemption mechanism "exists to deter malicious behavior by ensuring users cannot transfer their voting tokens until after the proposal has been resolved", which also indicates that the voting tokens are meant to be transferrable between users.

When the voting tokens are transferrable, one user can first use her or his votes to call the following `endorseProposal` function to endorse a proposal and then transfer these votes to another user. The other user can use these votes to endorse the same proposal again afterwards. Because of the double-endorsement, the `(totalEndorsementsForProposal[proposalId_] * 100) < VOTES.totalSupply() * ENDORSEMENT_THRESHOLD` condition can become true so the proposal can be activated by calling the `activateProposal` function below. However, the proposal should only be endorsed with these same votes once and should not be able to be activated if it could not satisify `(totalEndorsementsForProposal[proposalId_] * 100) < VOTES.totalSupply() * ENDORSEMENT_THRESHOLD` with these votes being used once.

https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Governance.sol#L180-L201

```
    function endorseProposal(uint256 proposalId_) external {
        uint256 userVotes = VOTES.balanceOf(msg.sender);

        if (proposalId_ == 0) {
            revert CannotEndorseNullProposal();
        }

        Instruction[] memory instructions = INSTR.getInstructions(proposalId_);
        if (instructions.length == 0) {
            revert CannotEndorseInvalidProposal();
        }

        // undo any previous endorsement the user made on these instructions
        uint256 previousEndorsement = userEndorsementsForProposal[proposalId_][msg.sende
        totalEndorsementsForProposal[proposalId_] -= previousEndorsement;

        // reapply user endorsements with most up-to-date votes
        userEndorsementsForProposal[proposalId_][msg.sender] = userVotes;
        totalEndorsementsForProposal[proposalId_] += userVotes;

        emit ProposalEndorsed(proposalId_, msg.sender, userVotes);
    }
```

https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Governance.sol#L205-L236

```
    function activateProposal(uint256 proposalId_) external {
        ProposalMetadata memory proposal = getProposalMetadata[proposalId_];

        if (msg.sender != proposal.submitter) {
            revert NotAuthorizedToActivateProposal();
        }

        if (block.timestamp > proposal.submissionTimestamp + ACTIVATION_DEADLINE) {
            revert SubmittedProposalHasExpired();
        }

        if (
            (totalEndorsementsForProposal[proposalId_] * 100) <
            VOTES.totalSupply() * ENDORSEMENT_THRESHOLD
        ) {
            revert NotEnoughEndorsementsToActivateProposal();
        }

        if (proposalHasBeenActivated[proposalId_] == true) {
            revert ProposalAlreadyActivated();
        }

        if (block.timestamp < activeProposal.activationTimestamp + GRACE_PERIOD) {
            revert ActiveProposalNotExpired();
        }

        activeProposal = ActivatedProposal(proposalId_, block.timestamp);

        proposalHasBeenActivated[proposalId_] = true;

        emit ProposalActivated(proposalId_, block.timestamp);
    }
```

Proof of Concept

Please append the following test in `src\test\policies\Governance.t.sol`. This test will pass to demonstrate the described scenario.

```
    function testScenario_UserEndorsesAfterReceivingTransferredVotes() public {
        _submitProposal();

        vm.prank(voter2);
        governance.endorseProposal(1);

        // to simulate calling gOHM's transfer function by voter2 for sending votes to v
        vm.prank(address(governance));
        VOTES.transferFrom(voter2, voter0, 200);

        // voter0 uses the votes previously owned by voter2 to endorse the proposal
        vm.prank(voter0);
        governance.endorseProposal(1);
        // the proposal is endorsed with 400 votes but only the 200 votes originally owr
        assertEq(governance.userEndorsementsForProposal(1, voter0), 200);
        assertEq(governance.userEndorsementsForProposal(1, voter2), 200);
        assertEq(governance.totalEndorsementsForProposal(1), 400);

        // At this moment, the proposal can be activated successfully.
        // However, if it is endorsed with only 200 votes, it cannot satisfy ENDORSEMENT
        vm.expectEmit(true, true, true, true);
        emit ProposalActivated(1, block.timestamp);

        vm.prank(voter1);
        governance.activateProposal(1);
    }
```

Tools Used

VSCode

Recommended Mitigation Steps

When calling `endorseProposal`, the user's votes can be locked by transferring these votes to the governance so the user cannot transfer these anymore to another user after the endorsement. An additional function can be added for reclaiming the endorsed votes back to the user and reducing the proposal's endorsed votes accordingly before the proposal is activated. After the proposal is activated, the endorsed votes should be counted as the voted votes.

fullyallocated (Olympus) acknowledged and commented:

> Taken from another issue:
>
> > This is true, and I appreciate the throughness of the explanation—it's hard to adjust endorsements based on the user's balance because there's no events/callbacks in solidity. We plan to use a staking vault where tokens are transfer locked and there's a warmup period + cooldown period to mitigate this issue.

[M-07] ENDORSED VOTES BY A USER DO NOT DECREASE AFTER THE USER'S VOTES ARE REVOKED

Submitted by rbserver

https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/VoterRegistration.sol#L53-L56
https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Governance.sol#L180-L201

The voter admin can call the following `revokeVotesFrom` function to revoke a user's votes, which also decreases the total supply of the votes, after the user endorses a proposal through calling the `endorseProposal` function below. Because `endorseProposal` can be called multiple times, the user has the incentive to call it for endorsing the proposal again with the new votes minted by the `issueVotesTo` function. However, after the user's votes are revoked, the user has no incentive to call `endorseProposal` again. Hence, the endorsed votes by the user for the proposal does not decrease after the user's votes are revoked. When determining whether the proposal can be activated or not, its old endorsed votes, which is not decreased, are compared against the new total supply of the votes, which is decreased because of the `revokeVotesFrom` call. As a result, the proposal is unreliably more likely to satisfy the condition for being activated.

https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/VoterRegistration.sol#L53-L56

```
    function revokeVotesFrom(address wallet_, uint256 amount_) external onlyRole("voter_
        // Revoke the votes in the VOTES module
        VOTES.burnFrom(wallet_, amount_);
    }
```

https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Governance.sol#L180-L201

```
    function endorseProposal(uint256 proposalId_) external {
        uint256 userVotes = VOTES.balanceOf(msg.sender);

        if (proposalId_ == 0) {
            revert CannotEndorseNullProposal();
        }

        Instruction[] memory instructions = INSTR.getInstructions(proposalId_);
        if (instructions.length == 0) {
            revert CannotEndorseInvalidProposal();
        }

        // undo any previous endorsement the user made on these instructions
        uint256 previousEndorsement = userEndorsementsForProposal[proposalId_][msg.sende
        totalEndorsementsForProposal[proposalId_] -= previousEndorsement;

        // reapply user endorsements with most up-to-date votes
        userEndorsementsForProposal[proposalId_][msg.sender] = userVotes;
        totalEndorsementsForProposal[proposalId_] += userVotes;

        emit ProposalEndorsed(proposalId_, msg.sender, userVotes);
    }
```

Proof of Concept

Please append the following test in `src\test\policies\Governance.t.sol`. This test will pass to demonstrate the described scenario.

```
    function testScenario_EndorsedVotesDoNotDecreaseAfterVotesAreRevoked() public {
        _submitProposal();

        // voter3 endorse the proposal
        vm.prank(voter3);
        governance.endorseProposal(1);

        assertEq(governance.userEndorsementsForProposal(1, voter3), 300);
        assertEq(governance.totalEndorsementsForProposal(1), 300);

        // to simulate calling VoterRegistration.revokeVotesFrom that burns voter3's vot
        vm.prank(godmode);
        VOTES.burnFrom(voter3, 300);

        // at this moment, voter3 has 0 votes
        assertEq(VOTES.balanceOf(voter3), 0);

        // however, the proposal is still endorsed with voter3's previous votes
        assertEq(governance.userEndorsementsForProposal(1, voter3), 300);
        assertEq(governance.totalEndorsementsForProposal(1), 300);
    }
```

Tools Used

VSCode

Recommended Mitigation Steps

When `revokeVotesFrom` is called during the time for endorsement, the corresponding votes that are previously endorsed for a proposal and are now revoked should be removed from the proposal's endorsed votes for the user. This ensures that the endorsed votes and the votes' total supply after the revocation are in sync for the proposal.

fullyallocated (Olympus) acknowledged and commented:

> This is true, and I appreciate the throughness of the explanation—it's hard to adjust endorsements based on the user's balance because there's no events/callbacks in solidity. We plan to use a staking vault where tokens are transfer locked and there's a warmup period + cooldown period to mitigate this issue.

[M-08] "TWAP" USED IS AN OBSERVATION-WEIGHTED-AVERAGE-PRICE, NOT A TIME-WEIGHTED ONE

Submitted by IllIllI

While users are incentivized to call the heartbeat, the incentive may be removed later, or it may be more profitable to use old prices, so users may not call the heartbeat during unfavorable prices, leading to the TWAP price being incorrect, and users getting the wrong price for their assets.

A similar case of an incomplete TWAP algorithm was found to be of Medium risk.

Proof of Concept

A TWAP is a Time-Weighted average price, but the algorithm below does not take into account the time between observations:

```
File: /src/modules/PRICE.sol   #1

134         // Calculate new moving average
135         if (currentPrice > earliestPrice) {
136             _movingAverage += (currentPrice - earliestPrice) / numObs;
137         } else {
138             _movingAverage -= (earliestPrice - currentPrice) / numObs;
139         }
140
141         // Push new observation into storage and store timestamp taken at
142         observations[nextObsIndex] = currentPrice;
143         lastObservationTime = uint48(block.timestamp);
144:        nextObsIndex = (nextObsIndex + 1) % numObs;
```

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/modules/PRICE.sol#L134-L144

While the `Heart` policy enforces an upper bound on how frequently updates are added to the average, there is no guarantee that users call `beat()` in a timely manner:

```
File: /src/policies/Heart.sol   #2

92      function beat() external nonReentrant {
93          if (!active) revert Heart_BeatStopped();
94:         if (block.timestamp < lastBeat + frequency()) revert Heart_OutOfCycle();
```

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Heart.sol#L92-L94

The incentive may be set to too low an amount:

```
File: /src/policies/Heart.sol   #3

    function setRewardTokenAndAmount(ERC20 token_, uint256 reward_)
        external
        onlyRole("heart_admin")
    {
        rewardToken = token_;
        reward = reward_;
        emit RewardUpdated(token_, reward_);
    }
```

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Heart.sol#L140-L147

Or users may find it more profitable to skip a particular update, or front-run an unfavorable update, with a transaction that trades assets at the old price

Recommended Mitigation Steps

Always call an internal version of `beat()` that doesn't revert, in functions that swap user assets. The code should also track the timestamps of when each `beat()` is called, and include the amount of time that has passed since the last beat, in the TWAP calculation

Oighty (Olympus) disagreed with severity and commented:

> The referenced issue is a bit different than our use case since we will be using a much longer duration moving average. The goal is to get an approximate moving average over a certain period of time (e.g. 120 days) vs. an exact number since, as you say, the time of each observation cannot be guaranteed to be at a specific time. We believe that using a long duration with a sufficient number of observations will make this value close enough to the true value it is approximating, and prevents actors from manipulating the value by waiting to provide a specific value (1 out of ~360 obs doesn't move the needle). The use of the "TWAP" term may be semantically inaccurate.
>
> As for not guaranteeing that the update will be called or issues with several observations close to each other, see comments on #405 and #79.
>
> The mitigations suggested do not seem to provide a solution that improves the system. Calling `beat()` on user actions would not have the observations roughly evenly spaced. Tracking timestamps is possible, but I don't see how it improves the data.

0xean (judge) commented:

> @Oighty - I think the warden is suggesting that the call to beat() in the user actions would do more to ensure that the "TWAP" stays up to date. If the call isn't past the correct period,

> it would just return and make no change (costing some amount of gas, ofc).
>
> I do think it may be worth considering, that way no user action can take place without the TWAP being as up to date as possible and no additional calls to the contract may be necessary if users are interacting with the contract frequently enough.
>
> While this is related to #79 - I think the points raised here and the mitigation is sufficiently different to warrant this issue to stand alone.

Oighty (Olympus) commented:

> That's a fair point. One issue with calling `beat` on user actions, e.g. `Operator.swap`, is that it would update the wall price that the user is swapping at. Therefore, the call could fail due to the slippage check. This could be confusing behavior and may have unintended consequences of DOS'ing the system. Additionally, the gas cost of `beat` is highly variable (sometimes up to 600k gas when opening a bond market) and would cause some users to unexpectedly pay a lot more gas for a swap.
>
> I'll discuss with the team, but I don't think the pros exceed the cons.

[M-09] ACTIVATEPROPOSAL() NEED TIME DELAY

Submitted by __141345__, also found by 0x1f8b, Trust, V_B, and zzzitron

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Governance.sol#L205-L262

There is no time lock or delay when activating a proposal, the previous one could be replaced immediately. In `vote()` call, a user might want to vote for the previous proposal, but if the `vote()` call and the `activateProposal()` is very close or even in the same block, it is quite possible that the user actually voted for another proposal without much knowledge of. A worse case is some malicious user watching the mempool, and front run a big vote favor/against the `activeProposal`, effectively influence the voting result.

These situations are not what the governance intends to deliver, and might also affect the results of 2 proposals.

Proof of Concept

`activateProposal()` can take effect right away, replacing the `activeProposal`. And `vote()` does not specify which `proposalId` to vote for, but the `activeProposal` could be different from last second.

src/policies/Governance.sol

```
    function activateProposal(uint256 proposalId_) external {
        ProposalMetadata memory proposal = getProposalMetadata[proposalId_];

        if (msg.sender != proposal.submitter) {
            revert NotAuthorizedToActivateProposal();
        }

        if (block.timestamp > proposal.submissionTimestamp + ACTIVATION_DEADLINE) {
            revert SubmittedProposalHasExpired();
        }

        if (
            (totalEndorsementsForProposal[proposalId_] * 100) <
            VOTES.totalSupply() * ENDORSEMENT_THRESHOLD
        ) {
            revert NotEnoughEndorsementsToActivateProposal();
        }

        if (proposalHasBeenActivated[proposalId_] == true) {
            revert ProposalAlreadyActivated();
        }

        if (block.timestamp < activeProposal.activationTimestamp + GRACE_PERIOD) {
            revert ActiveProposalNotExpired();
        }

        activeProposal = ActivatedProposal(proposalId_, block.timestamp);

        proposalHasBeenActivated[proposalId_] = true;

        emit ProposalActivated(proposalId_, block.timestamp);
    }

    function vote(bool for_) external {
        uint256 userVotes = VOTES.balanceOf(msg.sender);

        if (activeProposal.proposalId == 0) {
            revert NoActiveProposalDetected();
        }

        if (userVotesForProposal[activeProposal.proposalId][msg.sender] > 0) {
            revert UserAlreadyVoted();
        }

        if (for_) {
            yesVotesForProposal[activeProposal.proposalId] += userVotes;
        } else {
            noVotesForProposal[activeProposal.proposalId] += userVotes;
        }

        userVotesForProposal[activeProposal.proposalId][msg.sender] = userVotes;

        VOTES.transferFrom(msg.sender, address(this), userVotes);

        emit WalletVoted(activeProposal.proposalId, msg.sender, for_, userVotes);
    }
```

Recommended Mitigation Steps

Add time delay when activating a proposal, so that users can be aware of that and vote for the current one within the time window.

fullyallocated (Olympus) disputed and commented:

> This is a pretty unique edge case, I can acknowledge as QA.

0xean (judge) commented:

> I actually don't think its that unique in the case of on chain voting. Imagine a scenario where a user submits a vote with low gas amounts and it is not mined for days later and then the active proposal has changed. I am not sure why the `vote` function wouldn't take in the intended proposal ID.
>
> I am going to leave as medium severity as I do think this impacts the intended functionality of the protocol, but am willing to hear more from the sponsor on why they disagree.

[M-10] VOTED VOTES CANNOT CHANGE AFTER THE USER IS ISSUED NEW VOTES OR THE USER'S OLD VOTES ARE REVOKED DURING VOTING

Submitted by rbserver, also found by __141345__, cccz, csanuragjain, GalloDaSballo, Guardian, Lambda, m9800, and zzzitron

https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Governance.sol#L240-L262
https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/VoterRegistration.sol#L45-L48
https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/VoterRegistration.sol#L53-L56

A user can call the following `vote` function to vote for a proposal. During voting, the voter admin can still call the `issueVotesTo` and `revokeVotesFrom` functions below to issue new votes or revoke old votes for the user, which also changes the votes' total supply during the overall voting. Because each user can only call `vote` once for a proposal due to the `userVotesForProposal[activeProposal.proposalId][msg.sender] > 0` conditional check, the old voted votes, resulted from the `vote` call by the user, will be used to compare against the new total supply of the votes, resulted from the `issueVotesTo` and `revokeVotesFrom` calls during the overall voting, when determining whether the proposal can be executed or not. Because of this inconsistency, the result on whether the proposal can be executed might not be reliable.

https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Governance.sol#L240-L262

```
function vote(bool for_) external {
    uint256 userVotes = VOTES.balanceOf(msg.sender);

    if (activeProposal.proposalId == 0) {
        revert NoActiveProposalDetected();
    }

    if (userVotesForProposal[activeProposal.proposalId][msg.sender] > 0) {
        revert UserAlreadyVoted();
    }

    if (for_) {
        yesVotesForProposal[activeProposal.proposalId] += userVotes;
    } else {
        noVotesForProposal[activeProposal.proposalId] += userVotes;
    }

    userVotesForProposal[activeProposal.proposalId][msg.sender] = userVotes;

    VOTES.transferFrom(msg.sender, address(this), userVotes);

    emit WalletVoted(activeProposal.proposalId, msg.sender, for_, userVotes);
}
```

https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/VoterRegistration.sol#L45-L48

```
function issueVotesTo(address wallet_, uint256 amount_) external onlyRole("voter_adm
    // Issue the votes in the VOTES module
    VOTES.mintTo(wallet_, amount_);
}
```

https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/VoterRegistration.sol#L53-L56

```
function revokeVotesFrom(address wallet_, uint256 amount_) external onlyRole("voter_
    // Revoke the votes in the VOTES module
    VOTES.burnFrom(wallet_, amount_);
}
```

Proof of Concept

Please add the following code in `src\test\policies\Governance.t.sol`.

First, please add the following code for `stdError`.

```
import {Test, stdError} from "forge-std/Test.sol";    // @audit import stdError for test
```

Then, please append the following tests. These tests will pass to demonstrate the described scenarios.

```
    function testScenario_UserCannotVoteAgainWithNewlyMintedVotes() public {
        _createActiveProposal();

        // voter3 votes for the proposal
        vm.prank(voter3);
        governance.vote(true);

        assertEq(governance.yesVotesForProposal(1), 300);
        assertEq(governance.noVotesForProposal(1), 0);

        assertEq(governance.userVotesForProposal(1, voter3), 300);
        assertEq(VOTES.balanceOf(voter3), 0);
        assertEq(VOTES.balanceOf(address(governance)), 300);

        // to simulate calling VoterRegistration.issueVotesTo that mints votes to voter3
        vm.prank(godmode);
        VOTES.mintTo(voter3, 500);
        assertEq(VOTES.balanceOf(voter3), 500);

        // calling vote function again by voter3 reverts, which means that voter3 cannot
        vm.expectRevert(UserAlreadyVoted.selector);
        vm.prank(voter3);
        governance.vote(true);
    }
```

```
    function testScenario_RevokeVotesAfterUserFinishsOwnVoting() public {
        _createActiveProposal();

        // voter3 votes for the proposal
        vm.prank(voter3);
        governance.vote(true);

        assertEq(governance.yesVotesForProposal(1), 300);
        assertEq(governance.noVotesForProposal(1), 0);

        assertEq(governance.userVotesForProposal(1, voter3), 300);
        assertEq(VOTES.balanceOf(voter3), 0);
        assertEq(VOTES.balanceOf(address(governance)), 300);
        // To simulate calling VoterRegistration.revokeVotesFrom that burns voter3's vot
        // However, calling VOTES.burnFrom will revert due to arithmetic underflow.
        vm.prank(godmode);
        vm.expectRevert(stdError.arithmeticError);
        VOTES.burnFrom(voter3, 300);

        // the proposal is still voted with voter3's previous votes afterwards
        assertEq(governance.userVotesForProposal(1, voter3), 300);
        assertEq(VOTES.balanceOf(voter3), 0);
        assertEq(VOTES.balanceOf(address(governance)), 300);
    }
```

Tools Used

VSCode

Recommended Mitigation Steps

When `issueVotesTo` and `revokeVotesFrom` are called during voting, the corresponding votes need to be added to or removed from the proposal's voted votes for the user. Alternatively, `issueVotesTo` and `revokeVotesFrom` can be disabled when an active proposal exists.

fullyallocated (Olympus) confirmed and commented:

> This is the best written answer.
>
> Originally votes were locked so that users cannot constantly change their vote to manipulate the outcome but the warden makes a good point about how the quorum thresholds can be changed and the affects on how consensus is calculated.

[M-11] OLYMPUSGOVERNANCE: USERS CAN PREVENT THEIR VOTES FROM BEING REVOKED

Submitted by cccz, also found by zzzitron

In the VoterRegistration contract, voter_admin can call the revokeVotesFrom function to revoke the user's votes.

```
    function revokeVotesFrom(address wallet_, uint256 amount_) external onlyRole("voter_
        // Revoke the votes in the VOTES module
        VOTES.burnFrom(wallet_, amount_);
    }
```

But there is a way for users to prevent their votes from being revoked by voter*admin.
In the OlympusGovernance contract, the user can call the vote function to vote for the activeProposal, and then call the reclaimVotes function to reclaim his votes.
When the vote function is called, VOTES are sent to the OlympusGovernance contract and recorded using the userVotesForProposal variable. When the reclaimVotes function is called, the VOTES recorded in the userVotesForProposal variable are sent back to the user.
This means that the user can* store *his VOTES tokens in userVotesForProposal.
The revokeVotesFrom function cannot revoke the VOTES tokens recorded in userVotesForProposal and the reclaimVotes function can only be called by the user himself.
If the user calls the reclaimVotes function and vote function in one transaction, then his VOTES token balance will always be 0 (thus avoiding revocation of votes by voter*admin) and he will be able to vote.

Proof of Concept

https://github.com/code-423n4/2022-08-olympus/blob/2a0b515012b4a40076f6eac487f7816aafb8724a/src/policies/Governance.sol#L240-L262
https://github.com/code-423n4/2022-08-olympus/blob/2a0b515012b4a40076f6eac487f7816aafb8724a/src/policies/Governance.sol#L295-L313
https://github.com/code-423n4/2022-08-olympus/blob/2a0b515012b4a40076f6eac487f7816aafb8724a/src/policies/VoterRegistration.sol#L53-L56

Recommended Mitigation Steps

Consider allowing to call the reclaimVotes function to reclaim any user's vote, thus avoiding the user storing his VOTES tokens in userVotesForProposal

```
    function reclaimVotes(uint256 proposalId_, address user_) external {
        uint256 userVotes = userVotesForProposal[proposalId_][user_];

        if (userVotes == 0) {
            revert CannotReclaimZeroVotes();
        }

        if (proposalId_ == activeProposal.proposalId) {
            revert CannotReclaimTokensForActiveVote();
        }

        if (tokenClaimsForProposal[proposalId_][user_] == true) {
            revert VotingTokensAlreadyReclaimed();
        }

        tokenClaimsForProposal[proposalId_][user_] = true;

        VOTES.transferFrom(address(this), user_, userVotes);
    }
```

fullyallocated (Olympus) acknowledged and commented:

> This is true, we don't expect to use the voter admin in production, just to issue votes during internal testing period.

0xean (judge) decreased severity to Medium and commented:

> Downgrading to M severity as this does not lead to direct loss of user funds, but does highlight an issue with current contracts.

cccz (warden) commented:

> Consider the following scenarios. There are currently three users, A, B and C, in the system.
>
> 1. voter_admin minted 100 VOTEs for each of these three users
> 2. After a period of time, due to system upgrade or other reasons, the VOTEs of the users need to be revoked.
> 3. voter_admin revokes the VOTEs of users A and B respectively, but user C uses this vulnerability to prevent his VOTE from being revoked.
> 4. At this time, user C has all the VOTEs, and he can execute any proposal.

0xean (judge) commented:

> Okay, at this point I still believe a Medium issue, voter_admin as a mitigation could reissue votes to User A and B. Additionally User C will eventually have to reclaim these votes in order to vote on the next proposal. I am going to stick with Medium on this one. Appreciate the additional clarity.

cccz (warden) commented:

> @0xean - You are right, thanks for your attention.

[M-12] GRIEFING/DOS OF WITHDRAWALS BY EOAS FROM TREASURY (TRSRY) POSSIBLE

Submitted by minhtrng

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/TreasuryCustodian.sol#L5L67

Any withdrawals from the treasury by an approved EOA can be denied by a malicious actor that watches the mempool.

Proof of Concept

The function TreasuryCustodian.revokePolicyApprovals() doesnt provide sufficient checks for its intended purpose of "revoking a deactivated policy's approvals". As can be seen by the TODO labels, the issue has already been acknowledged by the team (regardless it is still an issue present in an in-scope contract). The only check performed is trying to call the isActive()-function on an address and interpret the returned value as boolean. Attempting to call this function on an EOA will not fail and return 0 (=false). Hence the condition to revert is not fulfilled and the amounts approved to withdraw will be set to 0.

Tools Used

IDE (Remix, VSCode)

Recommended Mitigation Steps

A partial but insufficient fix would be to check if the address passed to the function contains code and hence is not an EOA. A better approach might be to add a mapping(address => bool) of all addresses that have been active policies some time in the past to the kernel, something like this:

As a public variable in Kernel.sol `mapping(address => bool) public isRegisteredPolicy;`

in Kernel.activatePolicy(): `isRegisteredPolicy[address(policy_)] ) = true;`

and finally in TreasuryCustodian.revokePolicyApprovals():
`if(!kernel.isRegisteredPolicy(policy_) revert NotARegisteredPolicy`

ind-igo (Olympus) confirmed and commented:

> TODOs are outdated, I forgot to clear them ;(. But the points are taken. Code will be adjusted, but probably not the way from the recommendation. Instead will gate the function behind custodian role.

[M-13] MISSING CHECKS IN `KERNEL._DEACTIVATEPOLICY`

Submitted by enckrish

There are no checks to ascertain that the policy being removed is registered in the `Kernel`. Trying to remove a non-registered results in the policy registered at 0th index of `activePolicies` being removed.

Proof of Concept

https://github.com/code-423n4/2022-08-olympus/blob/main/src/Kernel.sol#L325

Recommended Mitigation Steps

Adding `require(activePolicies[idx] == policy_, "Unregistered policy");` will prevent this, where `idx = getPolicyIndex[policy_]`.

NOTE: The issue is less likely to happen as this is handled solely by the executor, but having safeguards in the contract is always better than relying on an external factor.

ind-igo (Olympus) confirmed, but disagreed with severity and commented:

> Confirmed. Should be lower risk or a QA issue.

0xean (judge) commented:

> @ind-igo - can you comment on why you think it should be QA vs Medium?
>
> ```
> — Med: Assets not at direct risk, but the function of the protocol or its
> availability could be impacted, or leak value with a hypothetical attack path
> with stated assumptions, but external requirements.
> ```
>
> I would expect this to impact the functionality of the protocol.

0ighty (Olympus) commented:

> This one seems on the fence to me. While accidentally unregistering a policy likely would affect the functionality of the protocol, it requires the executor to make a mistake. If the mistake is made, it’s easily remedied by re-registering the policy.

0xean (judge) commented:

> That makes sense, but there would be some amount of down time when this occurred. I think Medium is correct for this issue.

[M-14] THE GOVERNANCE SYSTEM CAN BE HELD HOSTAGE BY A MALICIOUS USER

Submitted by d3e4, also found by Aymen0909 and pedroais

With only `ENDORSEMENT_THRESHOLD`% (currently 20%) voting power, a malicious user can prevent any other proposal from being activated. While `ENDORSEMENT_THRESHOLD` is currently fairly high, it seems not higher than that it might not be used to hold the system hostage to extract far more funds.

Proof of Concept

Submit a dummy proposal, endorse it and then activate it. Now, no other proposal can be activated for a `GRACE_PERIOD`. When this time period is over, this procedure may be repeated, either immediately or just before any other proposal activation by front-running.

Recommended Mitigation Steps

Making sure `ENDORSEMENT_THRESHOLD` is at least 50% seems discouraging enough. Other more creative solutions should be possible. One might be to let the most endorsed proposal be activated, or restricting who can activate a proposal; anything that at least temporarily liberates the governance system so that the attacker is dissuaded from investing in this attack method.

fullyallocated (Olympus) acknowledged

[M-15] HEART WILL STOP IF ALL REWARDS ARE SWEPT

Submitted by GalloDaSballo, also found by cccz, itsmeSTYJ, and PwnPatrol

Rewards for Heart `beat` are sent via `_issueReward`

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Heart.sol#L110-L115

```
    function _issueReward(address to_) internal {
        rewardToken.safeTransfer(to_, reward);
        emit RewardIssued(to_, reward);
    }
```

The function doesn’t check for available tokens e.g. `min(reward, rewardToken.balanceOf(address(this)));`

In case of calling `withdrawUnspentRewards`

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Heart.sol#L149-L152

```
    /// @inheritdoc IHeart
    function withdrawUnspentRewards(ERC20 token_) external onlyRole("heart_admin") {
        token_.safeTransfer(msg.sender, token_.balanceOf(address(this)));
    }
```

Because the function withdraws the entire amount, the heart will stop until a caller incentive is deposited again.

While a profitable searches will stop calling the Heart without an incentive, allowing the heart to beat when no rewards are available is preferable to having it self-DOS until a DAO aligned caller donates `rewardToken` or the DAO deals with the lack of tokens.

Recommended Mitigation Steps

Add a check for available tokens `min(reward, rewardToken.balanceOf(address(this)));`

Oighty (Olympus) confirmed and commented:

> Agree based on the anti-DOS characteristics of using a min operation.

[M-16] INCONSISTANT PARAMETER REQUIREMENTS BETWEEN CONSTRUCTOR() AND SET() FUNCTIONS IN RANGE.SOL AND OPERATOR.SOL.

Submitted by hansfriese, also found by datapunk and itsmeSTYJ

Inconsistant parameter requirements between `constructor` and `Set() functions` in `RANGE.sol` and `Operator.sol`.

The contracts might work unexpectedly when the params are set improperly using `constructor()`.

Proof of Concept

- In `RANGE.sol`, setSpreads() and setThresholdFactor() has some requirements but constructor() doesn't check at all.

```
File: 2022-08-olympus\src\modules\RANGE.sol
    function setSpreads(uint256 cushionSpread_, uint256 wallSpread_) external
        // Confirm spreads are within allowed values
        if (
            wallSpread_ > 10000 ||
            wallSpread_ < 100 ||
            cushionSpread_ > 10000 ||
            cushionSpread_ < 100 ||
            cushionSpread_ > wallSpread_
        ) revert RANGE_InvalidParams();

        // Set spreads
        _range.wall.spread = wallSpread_;
        _range.cushion.spread = cushionSpread_;

        emit SpreadsChanged(wallSpread_, cushionSpread_);
    }
```

```
File: 2022-08-olympus\src\modules\RANGE.sol
    function setThresholdFactor(uint256 thresholdFactor_) external permissionec
        if (thresholdFactor_ > 10000 || thresholdFactor_ < 100) revert RANGE_Ir
        thresholdFactor = thresholdFactor_;

        emit ThresholdFactorChanged(thresholdFactor_);
    }

```

- In `Operator.sol`, setCushionFactor() checks the requirement but constructor() doesn't check it.

```
File: 2022-08-olympus\src\policies\Operator.sol
    function setCushionFactor(uint32 cushionFactor_) external onlyRole("operatc
        /// Confirm factor is within allowed values
        if (cushionFactor_ > 10000 || cushionFactor_ < 100) revert Operator_Inv

        /// Set factor
        _config.cushionFactor = cushionFactor_;

        emit CushionFactorChanged(cushionFactor_);
    }

```

Tools Used

Solidity Visual Developer of VSCode

Recommended Mitigation Steps

Recommend adding same validation for the parameters between `constructor()` and `Set()` functions.

Oighty (Olympus) disagreed with severity and commented:

> Agree that the constructor should validate these parameters, but it is only an issue if configured improperly.

0xean (judge) commented:

> While I am typically weary of marking input validations as medium severity, I do think in this case it's warranted as it directly leads to malfunctions at the protocol level and it seems that the sponsors thought it important enough to add the checks elsewhere. Hard call, but will award it at medium severity.

[M-17] NO CAP ON AMOUNT OF VOTES MEANS THE `VOTER_ADMIN` CAN GET ANY PROPOSAL TO PASS

Submitted by GalloDaSballo, also found by 0xNazgul, IllIIII, and rbserver

Because `VOTES` can be minted by `voter_admin`, and there is no cap on totalSupply, the `voter_admin` has the privileged ability to mint as many `VOTES` as they want in order to get any proposal to pass or veto it.

Proof of Concept - Veto

- Be `voter_admin`
- Mint XYZ tokens
- totalSupply is now higher so any proposal can be vetoed per this line

Proof of Concept - Approve

- Be `voter_admin`
- Mint XYZ tokens, where XYZ allows the `netVotes * 100 < VOTES.totalSupply() * EXECUTION_THRESHOLD` check to pass
- Mint `VOTES` to self
- Vote
- Proposal has passed

Recommended Mitigation Steps

Add a total supply cap to `VOTES`.

fullyallocated (Olympus) disputed and commented:

> This is possible but will not happen in a production environment because we're using this for internal testing.

0xean (judge) commented:

> Given the scope of the contracts the wardens were asked to review, I think this issue is valid. While I understand that the `voter_admin` is trusted, I don't think users expect the admin to be able to bypass any votes results in this manner.

[M-18] INCONSISTENCY IN STALENESS CHECKS BETWEEN OHM AND RESERVE TOKEN ORACLES

Submitted by okkothejawa, also found by datapunk, reassor, and Trust

https://github.com/code-423n4/2022-08-olympus/blob/main/src/modules/PRICE.sol#L165-L171

Price oracle may fail and revert due to the inconsistency in the staleness checks.

Proof of Concept

In the `getCurrentPrice()` of `PRICE.sol`, Chainlink oracles are used to get the price of OHM against a reserve token, and a staleness check is used to make sure the price oracles are reporting fresh data. Yet the freshness requirements are inconsistent, for OHM, `updatedAt` should be lower than current timestamp minus three times the observation frequency, while for the reserve price, it is required that `updatedAt` should be lower than current timestamp minus the observation frequency. Our understanding is that that frequency is multiplied by 3 so that there can be some meaningful room where price data is accepted, as the time frame of only observation frequency (multiplied by 1) may not be enough for the oracle to realistically update its data. (In other words, the frequency of new price information might be lower than the observation frequency, which is probably the case as third multiple is used for the OHM price). If this is the case, this inconsistency may lead to the `getCurrentPrice()` reverting as while third multiple of the observation frequency might give enough space for the first oracle, second oracle's first multiple of frequency time frame might not be enough and it couldn't pass the staleness check due to unrealistic expectation of freshness.

Tools Used

Manual review, talking with devs

Recommended Mitigation Steps

Change the line 171 to

```
            if (updatedAt < block.timestamp - 3 * uint256(observationFrequency))
```

like line 165.

Oighty (Olympus) confirmed and commented:

> This should indeed be the same. We will update to fix.

[M-19] TRSRY: REENTER FROM OLYMPUSTREASURY::REPAYLOAN TO OPERATOR::SWAP

Submitted by zzzitron

One can repay loan to the treasury with the value from the `Operator::swap`.

Condition:

- the reserve token in Operator has hook for sender (like ERC777)
- the debt is the same token as reserve

Proof of Concept

The below code snippet shows a part of proof of concept for reentrancy attack, which is based on `src/test/policies/Operator.t.sol`. The full test code can be found here, and git diff from the `Operator.t.sol`.

Let's say that the reserve token implements ERC777 with the hook for the sender (see weird erc20). If the attacker can take debt of the reserve currency for the attack contract `Reenterer`, the contract can call `OlympusTreasury::repayLoan` and in the middle of repay call `Operator::swap` function. The `swap` function will modify the reserve token balance of treasury and the amount the attacker swapped will be also be used for the `repayLoan`.

In the below example, the attacker has debt of 1e18, and repays 1e17. But since the `swap` function is called in the `repayLoan`, the debt is reduced 1e17 more then it should. And the swap happened as expected so the attack has the corresponding ohm token.

```
/// Mock to simulate the senders hook
/// for simplicity omitted the certain aspects like ERC1820 registry and etc.
contract MockERC777 is MockERC20 {
    constructor () MockERC20("ERC777", "777", 18) {}

    function transferFrom(address from, address to, uint256 amount) public override retu
        _callTokenToSend(from, to, amount);
        return super.transferFrom(from, to, amount);
        // _callTokenReceived(from, to, amount);
    }

    // simplified implementation for ERC777
    function _callTokenToSend(address from, address to, uint256 amount) private {
      if (from != address(0)) {
        IERC777Sender(from).tokensToSend(from, to, amount);
      }
    }
}

interface IERC777Sender {
  function tokensToSend(address from, address to, uint256 amount) external;
```

```
}

/// Concept for an attack contract
contract Reenterer is IERC777Sender {
  ERC20 public token;
  Operator public operator;
  bool public entered;

  constructor(address token_, Operator op_) {
    token = ERC20(token_);
    operator = op_;
  }

  function tokensToSend(address from, address to, uint256 amount) external override {
    if (!entered) {
    // call swap from reenter
    // which will manipulate the balance of treasury
      entered = true;
      operator.swap(token, 1e17, 0);
    }
  }

  function attack(OlympusTreasury treasury) public {
    // approve to the treasury
    token.approve(address(treasury), 1e18);
    token.approve(address(operator), 100* 1e18);

    // repayDebt of 1e17
    treasury.repayLoan(token, 1e17);
  }
}




/// the test
    function test_poc__reenter() public {
        vm.prank(guardian);
        operator.initialize();

      reserve.mint(address(reenterer), 1e18);
      assertEq(treasury.reserveDebt(reserve, address(reenterer)), 1e18);
      // start repayLoan
      reenterer.attack(treasury);
      // it should be 9 * 1e17 but it is 8 * 1e17
      assertEq(treasury.reserveDebt(reserve, address(reenterer)), 8*1e17);
    }
```

Cause

The `repayLoan`, in the line 110 below, calls the `safeTransferFrom`. The balance before and after are compared to determine how much of debt is paid. So, if the `safeTranferFrom` can modify the balance, the attacker can profit from it.

```
// OlympusTreasury::repayLoan

    function repayLoan(ERC20 token_, uint256 amount_) external nonReentrant {
        if (reserveDebt[token_][msg.sender] == 0) revert TRSRY_NoDebtOutstanding

        // Deposit from caller first (to handle nonstandard token transfers)
        uint256 prevBalance = token_.balanceOf(address(this));
        token_.safeTransferFrom(msg.sender, address(this), amount_);

        uint256 received = token_.balanceOf(address(this)) - prevBalance;
```

In the `swap` function, if the amount in token is reserve, the payment token to buy ohm will be paid to the treasury. It gives to an opportunity to modify the balance.

```
// Operator::swap

            /// Transfer reserves to treasury
            reserve.safeTransferFrom(msg.sender, address(TRSRY), amountIn_);
```

Although both of `Operator::swap` and `OlympusTreasury::repayLoan` have `nonReentrant` modifier, it does not prevent as they are two different contracts.

Tools Used

Foundry

Recommended Mitigation Steps

The deposit logic in the `OlympusTreasury::repayLoan` was trying to handle nonstandard tokens, such as fee-on-transfer. But by doing so introduced an attack vector for tokens with ERC777. If the reserve token should be decided in the governance, it should be clarified, which token standards can be used safely.

ind-igo (Olympus) confirmed and commented:

> Good report, although very low risk as the preconditions are extremely unlikely. Will take into account the suggestion by adding a comment to the function. Thank you.

0xean (judge) commented:

> I would probably downgrade to QA, but the warden does a good job of proving the point out with examples. Will leave as Medium.

[M-20] OPERATOR: IF WALLSPREAD IS 10000, OPERATE AND BEAT WILL REVERT AND PRICE INFORMATION CANNOT BE UPDATED ANYMORE

Submitted by zzzitron

The `beat` cannot be called anymore and price information will not be updated

Condition:

- the wallspread is set to 10000 (100%)
- lower wall is active (range.low.active==true)
- the price falls into the lower cushion (currentPrice < range.cushion.low.price && currentPrice > range.wall.low.price), therefore activates the lower bond market

Proof of Concept

The below proof of concept demonstrates that the `operate` will revert with 100% wallspread. The full test code can be found here as well as the diff from `Operator.t.sol`.

In the test, the wallspread was set to 10000, which is 100% (line 51). The price was set so that the lower market should be deployed (line 59). In the market deployment logic (`Operator::_activate`) will revert due to division by zero, and `operate` will fail.

Once this condition is met, the `operate` cannot be called and `Heart::beat` cannot be called as well, since the `Heart::beat` is calling `Operator::opearate` under the hood. As the result the price can never be updated. But other codes who uses the price information will not know that the price information is stale. If the upper wall is active and still have the capacity, one can swap from the upper wall using the stale information, which might cause some loss of funds.

```
function test_poc__lowCushionDeployWithWallspread10000Reverts() public {
    /// Initialize operator
    vm.prank(guardian);
    operator.initialize();

    /// if the wallspread is 10000 the operate might revert
    vm.prank(policy);
    operator.setSpreads(7000, 10000);

    /// Confirm that the cushion is not deployed
    assertTrue(!auctioneer.isLive(range.market(true)));

    /// Set price on mock oracle into the lower cushion
    /// given the lower wallspread is 10000
    /// when the lower market should be deployed the operate reverts
    price.setLastPrice(20 * 1e18);

    /// Trigger the operate function manually
```

```
            /// The operate will revert Error with "Division or modulo by 0"
            /// But I could not figure out to catch with `expectRevert`
            /// so just commenting out the assert
            // vm.prank(guardian);
            // /// vm.expectRevert(bytes("Division or module by 0"));   // this cannot catcl
            // operator.operate();
        }
```

Cause

The main cause is that the `RANGE::setSpreads` function fails to check for `wallSpread_ == 10000`. If the setter does not allow the wallSpread to be 100%, the price of the lower wall will not go to zero.

```
      // modules/RANGE.sol
250
251
252  242     function setSpreads(uint256 cushionSpread_, uint256 wallSpread_) external pe
253  243         // Confirm spreads are within allowed values
254  244         if (
255  245             wallSpread_ > 10000 ||
256  246             wallSpread_ < 100 ||
257  247             cushionSpread_ > 10000 ||
258  248             cushionSpread_ < 100 ||
259  249             cushionSpread_ > wallSpread_
260  250         ) revert RANGE_InvalidParams();
```

In the `RANGE::updatePrices`, the price of lower wall will be zero if the wallSpread is 100%. If the price of lower wall is zero, the `Operator::_activate` will fail for the lower cushion.

```
// policies/Operator.sol::_activate(bool high_)
// when high_ is false
421             uint256 invWallPrice = 10**(oracleDecimals * 2) / range.wall.low.price;

// modules/RANGE.sol::updatePrices
164         _range.wall.low.price = (movingAverage_ * (FACTOR_SCALE - wallSpread)) / FA(
```

Tools Used

Foundry

Recommended Mitigation Steps

Mitigation suggestion: line 245. Forbid wallSpread to be 100%.

```
    // modules/RANGE.sol


242     function setSpreads(uint256 cushionSpread_, uint256 wallSpread_) external pe
243         // Confirm spreads are within allowed values
244         if (
-245            wallSpread_ > 10000 ||
+               wallSpread_ >= 10000 ||
246             wallSpread_ < 100 ||
247             cushionSpread_ > 10000 ||
248             cushionSpread_ < 100 ||
249             cushionSpread_ > wallSpread_
250         ) revert RANGE_InvalidParams();
```

Oighty (Olympus) disagreed with severity and commented:

> This is indeed an edge case and we will update the value checks for the spread values to exclude `10000`. However, from a practical perspective, this is very unlikely to happen. If the goal is to set the lower wall to 0, then the system would just be disabled.

0xean (judge) commented:

> Given the warden does fully demonstrate the issue, I am going to award as Medium with the understanding that this is an extreme edge case.

[M-21] OLYMPUSGOVERNANCE - ACTIVE PROPOSAL DOES NOT EXPIRE

Submitted by reassor

Contract `OlympusGovernance` allows controlling protocol through on-chain governing. The issue is that once proposal is active it does not expire, which means that until the new proposal will be selected, anyone can vote on existing one and potentially execute it when it might cause harm to the protocol.

Scenario:

1. New proposal has been submited, endorsed and activated.
2. Users vote, but the quroum is not being achieved.
3. The proposal is active until new one is getting submitted.
4. 6 months elapses and the current active proposal might cause serious harm to the protocol (since it was created long time ago).
5. Malicious actor votes and executes proposal causing harm to the protocol.

Proof of Concept

`Governance.sol`:

- https://github.com/code-423n4/2022-08-olympus/blob/2a0b515012b4a40076f6eac487f7816aafb8724a/src/policies/Governance.sol#L265 L289

Tools Used

Manual Review / VSCode

Recommended Mitigation Steps

It is recommended to add expiration for the active proposal for example 2 weeks. After that time it should be possible to reject proposal and users should be able to reclaim VOTES tokens.

fullyallocated (Olympus) disputed

0xean (judge) commented:

> I believe the warden is simply stating that an active proposal stays active if not replaced. There is not expiration of a proposal once it becomes active, so theoretically if the governance process is inactive a very stale proposal could get executed.

[M-22] LOW MARKET BONDS/SWAPS NOT WORKING AFTER LOAN IS TAKEN FROM TREASURY

Submitted by immeas

https://github.com/code-423n4/2022-08-olympus/blob/main/src/modules/TRSRY.sol#L60

I am bordering between this being medium and low, but since this is, granted under very unlikely circumstances, hindering intended transfers to work I am submitting it as medium. That said, I don't think this scenario is very likely since it requires a trusted contract not part of initial release(? no contract in repo used a loan) to take a large loan from TRSRY.

Proof of Concept

This will cause test to fail on `TRANSFER_FAILED` due to TRSRY not having the tokens to transfer but `getReserveBalance` says it has, since capacity is determined based on non-existing tokens.

```
diff --git a/src/test/policies/Operator.t.sol b/src/test/policies/Operator.t.sol
```

```
index e09aec1..5c1e95f 100644
--- a/src/test/policies/Operator.t.sol
+++ b/src/test/policies/Operator.t.sol
@@ -26,6 +26,8 @@ import {OlympusMinter, OHM} from "modules/MINTR.sol";
 import {Operator} from "policies/Operator.sol";
 import {BondCallback} from "policies/BondCallback.sol";

+import {ModuleTestFixtureGenerator} from "test/lib/ModuleTestFixtureGenerator.sol";
+
 contract MockOhm is ERC20 {
     constructor(
         string memory _name,
@@ -45,6 +47,7 @@ contract MockOhm is ERC20 {
 // solhint-disable-next-line max-states-count
 contract OperatorTest is Test {
     using FullMath for uint256;
+    using ModuleTestFixtureGenerator for OlympusTreasury;

     UserFactory public userCreator;
     address internal alice;
@@ -53,6 +56,9 @@ contract OperatorTest is Test {
     address internal policy;
     address internal heart;

+    address public debtor;
+    address public godmode;
+
     RolesAuthority internal auth;
     BondAggregator internal aggregator;
     BondFixedTermTeller internal teller;
@@ -187,6 +193,18 @@ contract OperatorTest is Test {

         reserve.mint(address(treasury), testReserve * 100);

+        debtor = treasury.generateFunctionFixture(treasury.getLoan.selector);
+        godmode = treasury.generateGodmodeFixture(type(OlympusTreasury).name);
+
+        kernel.executeAction(Actions.ActivatePolicy, godmode);
+        kernel.executeAction(Actions.ActivatePolicy, debtor);
+
+        vm.prank(godmode);
+        treasury.setApprovalFor(debtor, reserve, testReserve * 100);
+
+        vm.prank(debtor);
+        treasury.getLoan(reserve,testReserve*100);
+
         // Approve the operator and bond teller for the tokens to swap
         vm.prank(alice);
         ohm.approve(address(operator), testOhm * 20);
```

Same is applicable for low market bonds since they are created based on the same capacity.

Tools Used

vs code + tests

Recommended Mitigation Steps

Determine capacity from actual tokens held by treasury.

ind-igo (Olympus) confirmed and commented:

> Acknowledged. Will add a reserve requirement check inside the TRSRY's debt functions, which we can expand with a policy to rebalance if out of balance on a heartbeat.

[M-23] TREASURY MODULE IS VULNERABLE TO CROSS-CONTRACT REENTRANCY

Submitted by Czar102

https://github.com/code-423n4/2022-08-olympus/blob/main/src/modules/TRSRY.sol#L108-L112

An attacker can pay back their loan to the treasury module with protocol-owned tokens. This will cause their loan to decrease despite the protocol won't be given funds for it.

Proof of Concept

The code first measures the number of tokens in the treasury, then transfers an amount to the contract and checks the change it caused. This is put behind a nonReentrant modifier so that one can't use the same balance change to pay back multiple parts of (potentially) multiple loans.

The problem arises when the treasury doesn't only claim tokens from paying back loans, but also claims protocol revenue. Since, an attacker can gain execution in the moment the funds are pulled to the treasury to trigger any function that grants treasury this type of tokens (collects protocol revenue). The contract will count these tokens as paying back one's loan since this happened between balance measurements.

Recommended Mitigation Steps

Add a function used to pull a token to the contract and mark it nonReentrant. Any transfer of tokens to the treasury should be done through that function.

ind-igo (Olympus) commented:

> I am confused by this submission. Need more information.

ind-igo (Olympus) confirmed and commented:

> Spoke with Czar, solution for minimal change is adding `received = min(received, amount_);`. Confirming issue.

[M-24] [NAZ-M1] CHAINLINK'S LATESTROUNDDATA MIGHT RETURN STALE RESULTS

Submitted by 0xNazgul, also found by __141345__, 0x1f8b, ak1, brgltd, cccz, csanuragjain, Dravee, Guardian, hyh, IllIIIIIIlI, itsmeSTYJ, Jujic, Lambda, pashov, peachtea, rbserver, reassor, Sm4rty, TomJ, and zzzitron

https://github.com/code-423n4/2022-08-olympus/blob/main/src/modules/PRICE.sol#L161
https://github.com/code-423n4/2022-08-olympus/blob/main/src/modules/PRICE.sol#L170

Across these contracts, you are using Chainlink's `latestRoundData` API, but there is only a check on `updatedAt`. This could lead to stale prices according to the Chainlink documentation:

- Historical Price data
- Checking Your returned answers

The result of `latestRoundData` API will be used across various functions, therefore, a stale price from Chainlink can lead to loss of funds to end-users.

Recommended Mitigation Steps

Consider adding the missing checks for stale data.

For example:

```
    (uint80 roundID ,answer,, uint256 timestamp, uint80 answeredInRound) = AggregatorV3Inter

    require(answer > 0, "Chainlink price <= 0");
    require(answeredInRound >= roundID, "Stale price");
    require(timestamp != 0, "Round not complete");
```

Oighty (Olympus) confirmed and commented:

> Agree. We'll add the additional checks.

[M-25] MOVING AVERAGE PRECISION IS LOST

Submitted by hyh, also found by CertoraInc, d3e4, and rbserver

Now the precision is lost in moving average calculations as the difference is calculated separately and added each time, while it typically can be small enough to lose precision in the division involved.

For example, `10000` moves of `990` size, `numObservations = 1000`. This will yield `0` on each update, while must yield `9900` increase in the moving average.

Proof of Concept

Moving average is calculated with the addition of the difference:

https://github.com/code-423n4/2022-08-olympus/blob/2a0b515012b4a40076f6eac487f7816aafb8724a/src/modules/PRICE.sol#L134-L139

```
        // Calculate new moving average
        if (currentPrice > earliestPrice) {
            _movingAverage += (currentPrice - earliestPrice) / numObs;
        } else {
            _movingAverage -= (earliestPrice - currentPrice) / numObs;
        }
```

`/ numObs` can lose precision as `currentPrice - earliestPrice` is usually small.

It is returned on request as is:

https://github.com/code-423n4/2022-08-olympus/blob/2a0b515012b4a40076f6eac487f7816aafb8724a/src/modules/PRICE.sol#L189-L193

```
    /// @notice Get the moving average of OHM in the Reserve asset over the defined winc
    function getMovingAverage() external view returns (uint256) {
        if (!initialized) revert Price_NotInitialized();
        return _movingAverage;
    }
```

Recommended Mitigation Steps

Consider storing the cumulative `sum`, while returning `sum / numObs` on request:

https://github.com/code-423n4/2022-08-olympus/blob/2a0b515012b4a40076f6eac487f7816aafb8724a/src/modules/PRICE.sol#L189-L193

```
    /// @notice Get the moving average of OHM in the Reserve asset over the defined winc
    function getMovingAverage() external view returns (uint256) {
        if (!initialized) revert Price_NotInitialized();
-       return _movingAverage;
+       return _movingAverage / numObservations;
    }
```

Oighty (Olympus) disagreed with severity and commented:

> Keeping track of the observations as a sum and then dividing is a good suggestion. The price values have 18 decimals and the max discrepancy introduced is very small (10**-15) with expected parameter ranges. The potential risk to the protocol seems low though.

hyh (warden) commented:

> Please notice that discrepancy here is unbounded, i.e. the logic itself does not have any max discrepancy, the divergence between fact and recorded value can pile up over time without a limit.
>
> If you do imply that markets should behave in some way that minuses be matched with pluses, then I must say that they really shouldn't.

Oighty (Olympus) confirmed

0xean (judge) commented:

> Debating between QA and Medium on this one. I am going to award it as medium because there is a potential to leak some value due to this imprecision compounding over time.

[M-26] CUSHION BOND MARKETS ARE OPENED AT WALL PRICE RATHER THAN CURRENT PRICE

Submitted by 0x52

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Operator.sol#L363-L469

Incorrect initial bond market price.

Proof of Concept

```
uint256 initialPrice = range.wall.high.price.mulDiv(bondScale, oracleScale);

uint256 initialPrice = invWallPrice.mulDiv(bondScale, oracleScale);
```

In the above lines the initial prices are set to the wall price rather than the current price as indicated in documentation.

Recommended Mitigation Steps

Initial price should be updated to open bond market at current price rather than wall price.

Oighty (Olympus) disagreed with severity and commented:

> This is more of a design decision than a bug. However, we did make this change in the code prior to the audit (it didn't get reflected in the repo). @ind-igo not sure how you want to handle.

0xean (judge) commented:

> Going to award as Medium assuming no additional input from sponsor on the topic.

Oighty (Olympus) commented:

> It does deviate from the spec so I guess that's appropriate. The system actually would work as-is, but would be less responsive to price movements into the cushions since the bond market would have to decay (which requires waiting) to reach the current market price vs. instantly providing a buy/sell at the current price.

[M-27] UNEXECUTABLE PROPOSALS WHEN `ACTIONS.MIGRATEKERNEL` IS NOT LAST INSTRUCTION

Submitted by Lambda

https://github.com/code-423n4/2022-08-olympus/blob/549b96bcf8b97807738572605f6b1e26b33ef411/src/modules/INSTR.sol#L61

In `INSTR.sol`, it is correctly checked that a `ChangeExecutor` instruction only occurs at the last position to avoid situations where the other instructions are deemed as invalid.
However, the same problem can occur for `MigrateKernel`. For instance, let's say we have a `MigrateKernel` followed by a `DeactivatePolicy` action. The `MigrateKernel` action will change the value of `kernel` within the policy. The `DeactivatePolicy` action tries to call `setActiveStatus` on the policy. However, this has a `onlyKernel` modifier and the call will therefore fail when it is done after the value of `kernel` was changed.

Recommended Mitigation Steps

Perform the same check for `MigrateKernel`.

fullyallocated (Olympus) confirmed and commented:

> Thank you; good catch

[M-28] ACTIVATING SAME POLICY MULTIPLE TIMES IN KERNEL POSSIBLE

Submitted by Lambda, also found by enckrish

https://github.com/code-423n4/2022-08-olympus/blob/549b96bcf8b97807738572605f6b1e26b33ef411/src/Kernel.sol#L296

To check that an already active policy is not added a second time, `isActive()` is called on the policy. However, `policy` could be a malicious contract that always returns `false` for `isActive()`. In such a scenario, it would be possible to activate the policy multiple times for the same Kernel. This would break uniqueness invariants such that `_deactivatePolicy()` no longer works. However, it could also be used for a DoS attack: As `_reconfigurePolicies` and `_migrateKernel` iterate over those lists that now contain duplicates, they could run out of gas if a policy is activated enough times.

Recommended Mitigation Steps

Check `getPolicyIndex[policy_] != 0` instead of relying on a value of an untrusted contract.

0xLienid (Olympus) commented:

> @ind-igo a few other submissions also mention problems with over-reliance on policy.isActive (i.e. #368). Might be worth mitigating with the suggested step here or the check on activePolicies[index] like 368 mentions.

ind-igo (Olympus) commented:

> Dupe of #368

0xean (judge) commented:

> I think this is separate from #368 which is about a policy deactivating that isn't already active.
>
> I am a bit skeptical at the impact statement currently, but it does seem like protocol functionality does end up in a bad state with the typical policy lifecycle here. Will award as Medium unless Sponsor wants to provide some additional reasoning as to a downgrade.

ind-igo (Olympus) commented:

> While the issue is slightly different from #368, the solution is the exact same. The remediation has the new checks to prevent both of these issues.

[M-29] TRSRY SUSCEPTIBLE TO LOAN / WITHDRAW CONFUSION

Submitted by Trust, also found by 0xSky, datapunk, and tonisives

https://github.com/code-423n4/2022-08-olympus/blob/main/src/modules/TRSRY.sol#L64-L102

Treasury allocates approvals in the withdrawApproval mapping which is set via setApprovalFor(). In both withdrawReserves() and in getLoan(), _checkApproval() is used to verify user has enough approval and subtracts the withdraw / loan amount. Therefore, there is no differentiation in validation between loan approval and withdraw approval. Policies which will use getLoan() (currently none) can simply withdraw the tokens without bookkeeping it as a loan.

Proof of Concept

1. Policy P has getLoan permission
2. setApprovalFor(policy, token, amount) was called to grant P permission to loan amount
3. P calls withdrawReserves(address, token, amount) and directly withdraws the funds without registering as loan

Recommended Mitigation Steps

A separate mapping called loanApproval should be implemented, and setLoanApprovalFor() will set it, getLoan() will reduce loanApproval balance.

ind-igo (Olympus) confirmed, but disagreed with severity and commented:

> Confirmed. Good suggestion. Would put as low risk though.

0xean (judge) commented:

> Currently thinking Medium is appropriate for this issue, but will circle back on it.

0xean (judge) commented:

> See #293 for a possible vector in which this could lead to loss of funds. Going to leave as Medium.

[M-30] `HEART::BEAT()` COULD BE CALLED SEVERAL TIMES IN ONE BLOCK IF NO ONE CALLED IT FOR A SOME TIME

Submitted by rvierdiiev, also found by datapunk, devtooligan, itsmeSTYJ, Jeiwan, Lambda, Trust, and zzzitron

`beat()` function is allowed to be called by anyone once in `frequency()` period. The purpose of it is to update the prices and do another operations related to bond market. User who ran it are rewarded. There is no need to run this function more then 1 time in `frequency()` period. However if `beat()` was last time called more then `frequency()` time ago then user can execute `beat()` function `(block.timestamp - lastBeat)/frequency()` times in a row in same block and get rewards.

Proof of Concept

https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Heart.sol#L92
https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Heart.sol#L103

Recommended Mitigation Steps

https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Heart.sol#L103
Change this line to `lastBeat = block.timestamp - (block.timestamp - lastBeat) % frequency();`
So no matter how much time the `beat()` was not called, it is possible to call it only once per `frequency()`.

Oighty (Olympus) confirmed and commented:

> See comment on #405. This approach actually solves both of our issues though.

0xean (judge) commented:

> Going to use this issue as the primary since the solution is elegant and solves the problem.

[M-31] PROTOCOL'S WALLS / CUSHION BONDS REMAIN ACTIVE EVEN IF HEART IS NOT BEATING

Submitted by Trust

https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Operator.sol#L188-L191
https://github.com/code-423n4/2022-08-olympus/blob/main/src/policies/Operator.sol#L272
https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Operator.sol#L346

The Walls of the RBS mechanism offer zero slippage swaps at the high and low of the moving average spread. The capacity to be swapped at these prices is usually very large, so it must make sure to only be enabled when the prices are guaranteed to be synced. However, there is no such check. If beat() is not called for some time, meaning we cannot determine if the current spread is legit, swap() still operates as usual.

Impact

The worst case scenario is that the wall is swapping at a losing price, meaning they can be immediately drained via arbitrage bot.

Proof of concept

```
1. Price is X at the start
2. Oracle stops updating for some reason / no one calls beat()
3. Price drops to Y , where Y < low wall centered around X
4. Attacker can perform arbitrage by buying Ohm at external markets at Y and selling Ohm
```

Recommended mitigation steps:

Change modifier onlyWhileActive to add a check for beat out of sync:

```
if (block.timestamp > lastBeat + SYNC_THRESHOLD * frequency())
```

Oighty (Olympus) confirmed and commented:

> This is an interesting unintended consequence of a bad price feed or other `beat()` issue. Your suggested update makes sense, but we will probably adjust slightly to only manage the bad data threshold in one place.
>
> After originally looking at it, I thought a try/catch block on the external call to `PRICE.getLastPrice()` in `Operator.operate()` would work, but it doesn't handle cases where `operate()` isn't reached by the `beat()` function.

[M-32] ADMIN CANNOT BE CHANGED TO EOA AFTER DEPLOYMENT

Submitted by Jeiwan, also found by datapunk

After contracts are deployed and initialized, the admin address in `Kernel` contract can only be set to a contract. Granting and revoking roles will be possible to do only via a contract, which looks like an unintended behavior since these operations cannot be performed via governance (the governance contract is designed to be the only executor).

Proof of Concept

Admin address can be changed to any address (EOA or contract) in the `executeAction` function in `Kernel`:
https://github.com/code-423n4/2022-08-olympus/blob/main/src/Kernel.sol#L252-L253

This piece explicitly allows EOA addresses since the other actions in the function (besides `ChangeExecutor`) are checked to have only a contract as the target (see `ensureContract` function calls in the other actions). This, and the fact that roles cannot be managed via governance, leads to the conclusion that an admin is designed to be an EOA.

However, in the `store` function in `INSTR`, action target can only be a contract:
https://github.com/code-423n4/2022-08-olympus/blob/main/src/modules/INSTR.sol#L52

After the contracts are deployed, `INSTR` will be the only contract that's allowed to call `Kernel.executeAction`:
https://github.com/code-423n4/2022-08-olympus/blob/main/src/scripts/Deploy.sol#L220

Thus, there will be no way to change admin to an EOA. If admin needs to be an EOA, the `INSTR` contract needs to be patched and re-deployed to allow non-contract targets.

Recommended Mitigation Steps

Allow EOA addresses as instruction targets or disallow non-contract admin addresses.

fullyallocated (Olympus) confirmed and commented:

> Nice find + writeup.

Low Risk and Non-Critical Issues

For this contest, 114 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by zzzitron received the top score from the judge.

The following wardens also submitted reports: c3phas, hyh, 0xNazgul, 0xNineDec, Jeiwan, Deivitto, Bahurum, cccz, rbserver, mics, Aymen0909, Rolezn, reassor, ignacio, oyc_109, 0xDjango, 0xSmartContract, shenwilly, rvierdiiev, Sm4rty, ReyAdmirado, Bnke0x0, Tomo, gogo, robee, fatherOfBlocks, ladboy233, erictee, ElKu, cRat1st0s, GalloDaSballo, Lambda, lukris02, tonisives, Ruhum, durianSausage, dipp, sikorico, IllIllI, 0xSky, pfapostol, Rohan16, DimSon, RaymondFam, Waze, devtooligan, 0x1f8b, nxrblsrpr, brgltd, delfin454000, BipinSah, bobirichman, datapunk, TomJ, martin, Ch_301, Chandr, ajtra, prasantgupta52, tnevler, rokinot, Guardian, w0Lfrum, CertoraInc, aviggiano, rajatbeladiya, yixxas, __141345__, csanuragjain, ak1, cryptphi, The_GUILD, 0x52, carlitox477, ch13fd357r0y3r, sorrynotsorry, PPrieditis, PwnPatrol, Chom, eierina, CodingNameKiki, StevenL, bin2chen, ret2basic, hansfriese, Funen, PaludoX0, Picodes, grGred, okkothejawa, Trust, natzuu, itsmeSTYJ, 0x040, d3e4, p_crypt0, 0xkatana, Margaret, 8olidity, LeoS, medikko, ne0n, *EthLedger, enckrish, Diraco, djxploit, JansenC, indijanc, CRYP70, apostle0x01, cloudjunky, MasterCookie, and 0x85102.*

SUMMARY

Risk	Title
L 01	Operator: incorrect accounting for fee-on-transfer reserve token
L-02	BondCallback: incorrect accounting if quoteToken is rebase token
L-03	PRICE: unsafe cast for `numObservations`
L-04	Operator: unsafe cast for decimals
L-05	BondCallback: operator is not set `constructor`
L-06	Operator: missing check for configParmas[0] (cushionFactor) in the constructor
L-07	Kernel: misplaced zero address check for `changeKernel`
L-08	BondCallback, Operator: upon module's upgrade, the token approval should be revoked
L-09	Heart: if the `_issueReward` fails the heart beat will revert
N-01	Kernel: missing zero address check for `executor` and `admin`
N-02	INSTR, Governance: upon module's upgrade, all instruction data should be carried over to the new modules
N-03	RANGE, PRICE: unused import of `FullMath`
N 04	PRICE: stale price

[L-01] OPERATOR: INCORRECT ACCOUNTING FOR FEE-ON-TRANSFER RESERVE TOKEN

- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Operator.sol#L330

If the reserve token is fee-to-transfer token and the user is buying ohm, the `Operator::swap` will incorrectly assume the `amountIn_` value is transferred, which fails to consider the fees. If the fee is rounded up, the attacker can purchase ohm without giving any assets to the treasury. It may not be profitable for the attacker, but it may cause devaluing of the ohm. However, the loss will be limited to the capacity.

```
// Operator::swap
// if(tokenIn_ == reserve) : buying ohm
329             /// Transfer reserves to treasury
330             reserve.safeTransferFrom(msg.sender, address(TRSRY), amountIn_);
```

[L-02] BONDCALLBACK: INCORRECT ACCOUNTING IF QUOTETOKEN IS REBASE TOKEN

- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/BondCallback.sol#L1

If the quoteToken is rebase token, the priorBalances may change due to rebasing or airdrop. It may result to an incorrect accounting. However, whether it is exploitable depends on the Bond market's logic.
With the current logic, it just checks whether the balance is increased more than the `inputAmount_`, so it is harder to exploit, compare to the alternative logic of using the difference in balances as the input amount. However, it also introduces the possibility of paying the users less than they deserve.

```
// Callback::callback
        // Check that quoteTokens were transferred prior to the call
        if (quoteToken.balanceOf(address(this)) < priorBalances[quoteToken] + inputA
            revert Callback_TokensNotReceived();
```

[L-03] PRICE: UNSAFE CAST FOR NUMOBSERVATIONS

- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/modules/PRICE.sol#L97
- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/modules/PRICE.sol#L249-L257
- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/modules/PRICE.sol#L281-L289

The `movingAverageDuration` and `observationFrequency` are uint48. So `movingAverageDuration / observationFrequency` may overflow when casted to uint32. In the below snippet, line 281, the array will be set based on the uint256 value, but the `numObservations` is casted down to uint32. It may result in different `numObservations` and the length of `observations` array. However, given the large numbers, the attempt to set such a large number as the parameters will likely to fail with "out of gas" error, since the length of the array `observations` is ridiculously large in this case. Yet, it is probably safe to set some upper limit for the `numObservations` or use safeCast.

```
// modules/PRICE::constructor
  97         numObservations = uint32(movingAverageDuration_ / observationFrequency_);

// modules/PRICE::changeObservationFrequency
```

```
        // Store blank observations array of new size
        observations = new uint256[](newObservations);

        // Set initialized to false and update state variables
        initialized = false;
        lastObservationTime = 0;
        _movingAverage = 0;
        nextObsIndex = 0;
        observationFrequency = observationFrequency_;
        numObservations = uint32(newObservations);
```

[L-04] OPERATOR: UNSAFE CAST FOR DECIMALS

- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Operator.sol#L372
- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Operator.sol#L427
- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Operator.sol#L431-L434
- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Operator.sol#L375-L379

In the `Operator::_activate` decimal values were casted to `int8` and `uint8` back and forth. Since there is no check, those values can potentially overflow/underflow. However, if it happens the exponent in the line 376 will like to revert due to too large numbers. Besides, if the price decimals are that big, this may not be the biggest problem to face.

```
// policies/Operator.sol::_activate

372             int8 scaleAdjustment = int8(ohmDecimals) - int8(reserveDecimals) + (pric

375             uint256 oracleScale = 10**uint8(int8(PRICE.decimals()) - priceDecimals);
376             uint256 bondScale = 10 **
377                 uint8(
378                     36 + scaleAdjustment + int8(reserveDecimals) - int8(ohmDecimals)
379                 );
```

[L-05] BONDCALLBACK: OPERATOR IS NOT SET CONSTRUCTOR

- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/BondCallback.sol#L3 L45

If the `operator` is not set, the `callback` function will revert so, it is crucial to set the `operator` before any operation. However, it was not set in the `constructor`, but should be set separately by calling `setOperator`.

[L-06] OPERATOR: MISSING CHECK FOR CONFIGPARMAS[0] (CUSHIONFACTOR) IN THE CONSTRUCTOR

- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Operator.sol#L92-L150
- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Operator.sol#L516-L524

The `Operator::constructor` does not check the condition of the `cushionFactor`. Below is the condition for the `cushionFactor` checked in the `Operator::setCushionFactor`.

```
// Operator::setCushionFactor

516     function setCushionFactor(uint32 cushionFactor_) external onlyRole("operator_pol
517         /// Confirm factor is within allowed values
518         if (cushionFactor_ > 10000 || cushionFactor_ < 100) revert Operator_InvalidP
519 
520         /// Set factor
521         _config.cushionFactor = cushionFactor_;
522 
523         emit CushionFactorChanged(cushionFactor_);
524     }
```

[L-07] KERNEL: MISPLACED ZERO ADDRESS CHECK FOR CHANGEKERNEL

- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/Kernel.sol#L76-L78
- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/Kernel.sol#L254-L257

Currently, the check for the `Kernel` to be a contract (also not to be the zero address), is in the current `Kernel` implementation. However, no modules and policies have the logic to ensure this as they inherit from `KernelAdapter`, which will just set the new kernel without a question. This will work well as long as the new Kernel has the similar logic to check the next Kernel's integrity. However, if the logic is forgotten, there is no other safe guard to ensure that the next kernel is not a zero address and is a contract. Since `Kernel` is absolutely needed for this system's functionality, there is no possible case that the Kernel should be the zero address. Therefore, it is probably safe to add the checking logic to the `KernelAdapter`, so every module and policy will check for the next Kernel. It costs more gas since the check is done multiple times, but still arguably it is worth the cost, since Kernel is core part of the system and it will not updated very often.

```
// KernelAdapter::changeKernel
 76     function changeKernel(Kernel newKernel_) external onlyKernel {
 77         kernel = newKernel_;
 78     }

// Kernel::executeAction
254         } else if (action_ == Actions.MigrateKernel) {
255             ensureContract(target_);
256             _migrateKernel(Kernel(target_));
257         }
```

[L-08] BONDCALLBACK, OPERATOR: UPON MODULE'S UPGRADE, THE TOKEN APPROVAL SHOULD BE REVOKED

- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/BondCallback.sol#L5
- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Operator.sol#L167

The `BondCallback` and `Operator` approves ohm to the `MINTR` module in the `configureDependencies`. However, there is no logic to revoke those approvals (e.i. approve to zero). In the case of the `MINTR` has some bugs, it may be desirable to be able to revoke the approvals.

```
// Operator::configureDependencies
167         ohm.safeApprove(address(MINTR), type(uint256).max);
```

[L-09] HEART: IF THE ISSUEREWARD FAILS THE HEART BEAT WILL REVERT

- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Heart.sol#L106

If the `_issueReward` reverts, for example, because the token balance is too low, the `beat` will as well revert, due to the `safeTransfer`. One might consider not to revert even in the case the `_issueReward` reverts.

[N-01] KERNEL: MISSING ZERO ADDRESS CHECK FOR EXECUTOR AND ADMIN

- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/Kernel.sol#L250-L253

The `executor` and `admin` are not checked for the zero address when set by the `Kernel::executeAction`.

```
// Kernel::executeAction
250         } else if (action_ == Actions.ChangeExecutor) {
251             executor = target_;
252         } else if (action_ == Actions.ChangeAdmin) {
253             admin = target_;
```

[N-02] INSTR, GOVERNANCE: UPON MODULE'S UPGRADE, ALL INSTRUCTION DATA SHOULD BE CARRIED OVER TO THE NEW MODULES

- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Governance.sol#L167
- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/policies/Governance.sol#L187

The `Governance`'s logic will break if the `INSTR` module is upgraded to a new contract without having the same instructions data, since the `proposalId`'s the `Governance` is using are bound to the `INSTR` module.

[N-03] RANGE, PRICE: UNUSED IMPORT OF FULLMATH

- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/modules/RANGE.sol#L18
- https://github.com/code-423n4/2022-08-olympus/blob/b5e139d732eb4c07102f149fb9426d356af617aa/src/modules/PRICE.sol#L23

The modules `RANGE` and `PRICE` imports `FullMath`, but it is not used.

```
// modules/PRICE.sol
 22 contract OlympusPrice is Module {
 23     using FullMath for uint256;
```

[N-04] PRICE: STALE PRICE

There is no indicator whether the price information is up-to-date. If the price information is not properly updated, the other contracts will keep using the data resulting in incorrect prices for swap.

Gas Optimizations

For this contest, 91 reports were submitted by wardens detailing gas optimizations. The report highlighted below by pfapostol received the top score from the judge.

The following wardens also submitted reports: 0xSmartContract, 0x1f8b, LeoS, Tomo, 0xNazgul, m_Rassska, Aymen0909, ReyAdmirado, gogo, lllllll, ret2basic, c3phas, ajtra, JC, __141345__, TomJ, ignacio, Deivitto, grGred, Rolezn, 0xkatana, Sm4rty, brgltd, oyc_109, robee, 0x040, Bnke0x0, exolorkistis, ladboy233, durianSausage, erictee, martin, carlitox477, zishansami, Rohan16, rbserver, Dionysus, tnevler, GalloDaSballo, StevenL, fatherOfBlocks, CertoraInc, ch0bu, jag, ElKu, lukris02, 0xDjango, medikko, Noah3o6, Saintcode_, CodingNameKiki, Ruhum, chrisdior4, Amithuddar, cccz, bobirichman, cRat1st0s, Guardian, 0x85102, Shishigami, Metatron, RaymondFam, 0xNineDec, Waze, RoiEvenHaim, Chandr, apostle0x01, Funen, d3e4, natzuu, aviggiano, djxploit, peiw, JansenC, karanctf, kris, simon135, Tagir2003, Diraco, delfin454000, SooYa, rokinot, ne0n, rvierdiiev, The_GUILD, newfork01, Jeiwan, sikorico, Fitraldys, and hyh.

SUMMARY

Gas savings are estimated using the gas report of existing `FORGE_GAS_REPORT=true forge test` tests (the sum of all deployment costs and the sum of the costs of calling all methods) and may vary depending on the implementation of the fix. I keep my version of the fix for each finding and can provide them if you need.
Some optimizations (mostly logical) cannot be scored with a exact gas quantity.

	Issue	Instances	Estimated gas(deployments)	Estimated gas(method call)
G 01	Replace `modifier` with `function`	6	460 154	-
G-02	`storage` pointer to a structure is cheaper than copying each value of the structure into `memory`, same for `array` and `mapping`	7	188 639	5 032
G-03	Using `private` rather than `public` for constants, saves gas	8	45 857	308
G-04	Use elementary types or a user-defined `type` instead of a `struct` that has only one member	1	30 714	1 037
G-05	State variables should be cached in stack variables rather than re-reading them from storage	7	24 021	614
G-06	Using bools for storage incurs overhead	6	23 611	4 485
G-07	State variables can be packed into fewer storage slots	3	23 292	1 711
G-08	Expressions that cannot be overflowed can be unchecked	5	23 016	-
G-09	Increment optimization	18	↓	↓
G-09.1	Prefix increments are cheaper than postfix increments, especially when it's used in for-loops	3	400	-
G 09.2	`<x> = <x> + 1` even more efficient than pre increment	18	14 217	-
G-10	Use named `returns` for local variables where it is possible	3	5 400	-
G-11	`x = x + y` is cheaper than `x += y;`	6	5 000	-
G-12	Deleting a struct is cheaper than creating a new struct with null values.	1	4 207	-
G-13	Don't compare boolean expressions to boolean literals	2	1 607	-
G 14	`revert` operator should be in the code as early as reasonably possible	3	200	1 559+
G 15	Duplicated require()/revert() checks should be refactored to a modifier or function	4	-	8 111

Total: 83 instances over 15 issues

[G-01] REPLACE `MODIFIER` WITH `FUNCTION` (6 INSTANCES)

Modifiers make code more elegant, but cost more than normal functions.

Deployment Gas Saved: 460 154

All modifiers except `permissioned` due to unresolved error flow.

- src/Kernel.sol:70-73, 119-123, 223-232

```
70      modifier onlyKernel() {
```

```
71          if (msg.sender != address(kernel)) revert KernelAdapter_OnlyKernel(msg.sende
72          _;
73      }
...
119     modifier onlyRole(bytes32 role_) {
120         Role role = toRole(role_);
121         if (!kernel.hasRole(msg.sender, role)) revert Policy_OnlyRole(role);
122         _;
123     }
...
223     modifier onlyExecutor() {
224         if (msg.sender != executor) revert Kernel_OnlyExecutor(msg.sender);
225         _;
226     }
227
228     /// @notice Modifier to check if caller is the roles admin.
229     modifier onlyAdmin() {
230         if (msg.sender != admin) revert Kernel_OnlyAdmin(msg.sender);
231         _;
232     }
```

- src/policies/Operator.sol:188-191

```
188     modifier onlyWhileActive() {
189         if (!active) revert Operator_Inactive();
190         _;
191     }
```

- src/modules/PRICE.sol

```
if (!initialized) revert Price_NotInitialized(); // @note 4 instances
```

[G-02] STORAGE POINTER TO A STRUCTURE IS CHEAPER THAN COPYING EACH VALUE OF THE STRUCTURE INTO MEMORY, SAME FOR ARRAY AND MAPPING (7 INSTANCES)

Deployment Gas Saved: 188 639
Method Call Gas Saved: 5 032

It may not be obvious, but every time you copy a storage `struct`/`array`/`mapping` to a `memory` variable, you are literally copying each member by reading it from `storage`, which is expensive. And when you use the `storage` keyword, you are just storing a pointer to the storage, which is much cheaper.

- src/Kernel.sol:379

```
379        Policy[] memory dependents = moduleDependents[keycode_];
```

fix(the same for others):

```
Policy[] storage dependents = moduleDependents[keycode_];
```

- src/policies/BondCallback.sol:179

```
179        uint256[2] memory marketAmounts = _amountsPerMarket[id_];
```

- src/policies/Governance.sol:206

```
206        ProposalMetadata memory proposal = getProposalMetadata[proposalId_];
```

- src/policies/Operator.sol:205-206, 384-385, 439-440, 666

```
205        /// Cache config in memory
206        Config memory config_ = _config;
...
384            /// Cache config struct to avoid multiple SLOADs
385            Config memory config_ = _config;
...
439            /// Cache config struct to avoid multiple SLOADs
440            Config memory config_ = _config;
...
666        Regen memory regen = _status.low;
```

[G-03] USING PRIVATE RATHER THAN PUBLIC FOR CONSTANTS, SAVES GAS (8 INSTANCES)

If needed, the value can be read from the verified contract source code. Savings are due to the compiler not having to create non-payable getter functions for deployment calldata, and not adding another entry to the method ID table

Deployment Gas Saved: 45 857
Method Call Gas Saved: 308

- src/policies/Governance.sol:119-137

```
/// @notice The amount of votes a proposer needs in order to submit a proposal as
/// @dev    This is set to 1% of the total supply.
uint256 public constant SUBMISSION_REQUIREMENT = 100;

/// @notice Amount of time a submitted proposal has to activate before it expires
uint256 public constant ACTIVATION_DEADLINE = 2 weeks;

/// @notice Amount of time an activated proposal must stay up before it can be re
uint256 public constant GRACE_PERIOD = 1 weeks;

/// @notice Endorsements required to activate a proposal as percentage of total s
uint256 public constant ENDORSEMENT_THRESHOLD = 20;

/// @notice Net votes required to execute a proposal on chain as a percentage of
uint256 public constant EXECUTION_THRESHOLD = 33;

/// @notice Required time for a proposal to be active before it can be executed.
/// @dev    This amount should be greater than 0 to prevent flash loan attacks.
uint256 public constant EXECUTION_TIMELOCK = 3 days;
```

- src/policies/Operator.sol:89

```
uint32 public constant FACTOR_SCALE = 1e4;
```

- src/modules/RANGE.sol:65

```
uint256 public constant FACTOR_SCALE = 1e4;
```

[G-04] USE ELEMENTARY TYPES OR A USER-DEFINED TYPE INSTEAD OF A STRUCT THAT HAS ONLY ONE MEMBER (1 INSTANCES)

Deployment Gas Saved: 30 714
Method Call Gas Saved: 1 037

- src/modules/RANGE.sol:33-35

```
    struct Line {
        uint256 price; // Price for the specified level
    }
```

[G-05] STATE VARIABLES SHOULD BE CACHED IN STACK VARIABLES RATHER THAN RE-READING THEM FROM STORAGE

Deployment Gas Saved: 24 021
Method Call Gas Saved: 614

SLOADs are expensive (100 gas after the 1st one) compared to MLOADs/MSTOREs (3 gas each). Storage values read multiple times should instead be cached in memory the first time (costing 1 SLOAD) and then read from this cache to avoid multiple SLOADs.

- src/policies/Heart.sol:112-113

```
        rewardToken.safeTransfer(to_, reward);
        emit RewardIssued(to_, reward);
```

fix:

```
uint256 reward = reward;
rewardToken.safeTransfer(to_, reward);
emit RewardIssued(to_, reward);
```

- src/policies/BondCallback.sol:68-75

```
68          Keycode TRSRY_KEYCODE = TRSRY.KEYCODE();
69          Keycode MINTR_KEYCODE = MINTR.KEYCODE();
70
71          requests = new Permissions[](4);
72          requests[0] = Permissions(TRSRY_KEYCODE, TRSRY.setApprovalFor.selector);
73          requests[1] = Permissions(TRSRY_KEYCODE, TRSRY.withdrawReserves.selector);
74          requests[2] = Permissions(MINTR_KEYCODE, MINTR.mintOhm.selector);
75          requests[3] = Permissions(MINTR_KEYCODE, MINTR.burnOhm.selector);
```

fix(similar for other policies):

```
    OlympusTreasury ltrsry = TRSRY;
    OlympusMinter lmintr = MINTR;
    Keycode TRSRY_KEYCODE = ltrsry.KEYCODE();
    Keycode MINTR_KEYCODE = lmintr.KEYCODE();

    requests = new Permissions[](4);

    requests[0] = Permissions(TRSRY_KEYCODE, ltrsry.setApprovalFor.selector);
    requests[1] = Permissions(TRSRY_KEYCODE, ltrsry.withdrawReserves.selector);
    requests[2] = Permissions(MINTR_KEYCODE, lmintr.mintOhm.selector);
    requests[3] = Permissions(MINTR_KEYCODE, lmintr.burnOhm.selector);
```

- src/policies/Governance.sol:77-79

```
77          requests = new Permissions[](2);
78          requests[0] = Permissions(INSTR.KEYCODE(), INSTR.store.selector);
79          requests[1] = Permissions(VOTES.KEYCODE(), VOTES.transferFrom.selector);
```

- src/policies/Operator.sol:172-185

```
172         Keycode RANGE_KEYCODE = RANGE.KEYCODE();
173         Keycode TRSRY_KEYCODE = TRSRY.KEYCODE();
174         Keycode MINTR_KEYCODE = MINTR.KEYCODE();
175
176         requests = new Permissions[](9);
177         requests[0] = Permissions(RANGE_KEYCODE, RANGE.updateCapacity.selector);
178         requests[1] = Permissions(RANGE_KEYCODE, RANGE.updateMarket.selector);
179         requests[2] = Permissions(RANGE_KEYCODE, RANGE.updatePrices.selector);
180         requests[3] = Permissions(RANGE_KEYCODE, RANGE.regenerate.selector);
181         requests[4] = Permissions(RANGE_KEYCODE, RANGE.setSpreads.selector);
182         requests[5] = Permissions(RANGE_KEYCODE, RANGE.setThresholdFactor.selector);
183         requests[6] = Permissions(TRSRY_KEYCODE, TRSRY.setApprovalFor.selector);
184         requests[7] = Permissions(MINTR_KEYCODE, MINTR.mintOhm.selector);
185         requests[8] = Permissions(MINTR_KEYCODE, MINTR.burnOhm.selector);
```

- src/policies/PriceConfig.sol:32-34

```
        permissions[0] = Permissions(PRICE.KEYCODE(), PRICE.initialize.selector);
        permissions[1] = Permissions(PRICE.KEYCODE(), PRICE.changeMovingAverageDuratic
        permissions[2] = Permissions(PRICE.KEYCODE(), PRICE.changeObservationFrequency
```

- src/policies/TreasuryCustodian.sol:35-39

```
        Keycode TRSRY_KEYCODE = TRSRY.KEYCODE();

        requests = new Permissions[](2);
        requests[0] = Permissions(TRSRY_KEYCODE, TRSRY.setApprovalFor.selector);
        requests[1] = Permissions(TRSRY_KEYCODE, TRSRY.setDebt.selector);
```

- src/policies/VoterRegistration.sol:33-35

```
        permissions = new Permissions[](2);
        permissions[0] = Permissions(VOTES.KEYCODE(), VOTES.mintTo.selector);
        permissions[1] = Permissions(VOTES.KEYCODE(), VOTES.burnFrom.selector);
```

[G-06] USING BOOLS FOR STORAGE INCURS OVERHEAD (6 INSTANCES)

Deployment Gas Saved: 23 611
Method Call Gas Saved: 4 485

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

Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas) for the extra SLOAD, and to avoid Gsset (20000 gas) when changing from 'false' to 'true', after having been 'true' in the past.

Important: This rule doesn't always work, sometimes a bool is packed with another variable in the same slot, sometimes it's packed into a struct, sometimes the optimizer makes bool more efficient. You can see the @note in the code for each case.

- src/Kernel.sol:181, 194, 197

```
181    mapping(Keycode => mapping(Policy => mapping(bytes4 => bool))) public modulePermi
...
194    mapping(address => mapping(Role => bool)) public hasRole; //@note D:-3016 M:2298
...
197    mapping(Role => bool) public isRole; //@note D:2407
```

- src/policies/Governance.sol:105, 117,

```
105    mapping(uint256 => bool) public proposalHasBeenActivated; //@note D:3007
...
117    mapping(uint256 => mapping(address => bool)) public tokenClaimsForProposal; //@nc
```

- src/modules/PRICE.sol:62

```
62     bool public initialized; //@note D:11813
```

Expensive method calls:

It's just to show which bool is better left in the code

- src/policies/Operator.sol

```
63     bool public initialized; //@note D:5808 M:-22036
...
66     bool public active; //@note D:-32775 M:-48896
```

- src/policies/Heart.sol

```
33     bool public active; //@note D:-382
```

- src/policies/BondCallback.sol

```
24     mapping(address => mapping(uint256 => bool)) public approvedMarkets; //@note D:-
```

- src/Kernel.sol

```
113    bool public isActive; //@note D:20923 M:-247184
```

[G-07] STATE VARIABLES CAN BE PACKED INTO FEWER STORAGE SLOTS (3 INSTANCES)

If variables occupying the same slot are both written the same function or by the constructor, avoids a separate Gsset (20000 gas). Reads of the variables can also be cheaper.

NOTE: one slot = 32 bytes

Deployment Gas Saved: 23 292
Method Call Gas Saved: 1 711

- src/policies/Heart.sol:32-48

uint256(32), address(20), bool(1)

```
32     /// @notice Status of the Heart, false = stopped, true = beating
33     bool public active; // @note put below _operator
34
35     /// @notice Timestamp of the last beat (UTC, in seconds)
36     uint256 public lastBeat;
37
38     /// @notice Reward for beating the Heart (in reward token decimals)
39     uint256 public reward;
40
41     /// @notice Reward token address that users are sent for beating the Heart
42     ERC20 public rewardToken;
43
44     // Modules
```

```
45      OlympusPrice internal PRICE;
46
47      // Policies
48      IOperator internal _operator;
```

fix:

```
uint256 public lastBeat;
uint256 public reward;
ERC20 public rewardToken;
OlympusPrice internal PRICE;
IOperator internal _operator;
bool public active;
```

- src/modules/PRICE.sol:31-65

NOTE: PRICE is Module, Module is KernelAdapter, so real first variable in PRICE is kernel from KernelAdapter

uint256(32), uint32(4), uint48(6), uint8(1), array(32), address(20), bool(1)

```
inherit Kernel public kernel;
...
31      /// @dev    Price feeds. Chainlink typically provides price feeds for an asset ir
32      AggregatorV2V3Interface internal immutable _ohmEthPriceFeed;
33      AggregatorV2V3Interface internal immutable _reserveEthPriceFeed;
34
35      /// @dev Moving average data
36      uint256 internal _movingAverage; /// See getMovingAverage()
37
38      /// @notice Array of price observations. Check nextObsIndex to determine latest c
39      /// @dev    Observations are stored in a ring buffer where the moving average is
40      ///         Observations can be cleared by changing the movingAverageDuration or
41      uint256[] public observations;
42
43      /// @notice Index of the next observation to make. The current value at this inde
44      uint32 public nextObsIndex;
45
46      /// @notice Number of observations used in the moving average calculation. Comput
47      uint32 public numObservations;
48
49      /// @notice Frequency (in seconds) that observations should be stored.
50      uint48 public observationFrequency;
51
52      /// @notice Duration (in seconds) over which the moving average is calculated.
53      uint48 public movingAverageDuration;
54
55      /// @notice Unix timestamp of last observation (in seconds).
56      uint48 public lastObservationTime;
57
58      /// @notice Number of decimals in the price values provided by the contract.
```

```
    uint8 public constant decimals = 18;

    /// @notice Whether the price module is initialized (and therefore active).
    bool public initialized;

    // Scale factor for converting prices, calculated from decimal values.
    uint256 internal immutable _scaleFactor;
```

fix:

```
uint48 public observationFrequency;
uint48 public movingAverageDuration;
AggregatorV2V3Interface internal immutable _ohmEthPriceFeed;
AggregatorV2V3Interface internal immutable _reserveEthPriceFeed;
uint256 internal _movingAverage; /// See getMovingAverage()
uint256[] public observations;
uint32 public nextObsIndex;
uint32 public numObservations;
uint48 public lastObservationTime;
uint8 public constant decimals = 18;
bool public initialized;
uint256 internal immutable _scaleFactor;
```

- src/policies/Operator.sol:58-89

uint32(4), uint8(1), address(20), bool(1)

```
    /// Operator variables, defined in the interface on the external getter functions
    Status internal _status;
    Config internal _config;

    /// @notice    Whether the Operator has been initialized
    bool public initialized;

    /// @notice    Whether the Operator is active
    bool public active;

    /// Modules
    OlympusPrice internal PRICE;
    OlympusRange internal RANGE;
    OlympusTreasury internal TRSRY;
    OlympusMinter internal MINTR;

    /// External contracts
    /// @notice     Auctioneer contract used for cushion bond market deployments
    IBondAuctioneer public auctioneer;
    /// @notice     Callback contract used for cushion bond market payouts
    IBondCallback public callback;

    /// Tokens
    /// @notice     OHM token contract
```

```
82      ERC20 public immutable ohm;
83      uint8 public immutable ohmDecimals;
84      /// @notice     Reserve token contract
85      ERC20 public immutable reserve;
86      uint8 public immutable reserveDecimals;
87
88      /// Constants
89      uint32 public constant FACTOR_SCALE = 1e4;
```

fix:

```
Status internal _status;
Config internal _config;
OlympusPrice internal PRICE;
OlympusRange internal RANGE;
OlympusTreasury internal TRSRY;
OlympusMinter internal MINTR;
IBondAuctioneer public auctioneer;
IBondCallback public callback;
bool public initialized;
bool public active;
ERC20 public immutable ohm;
uint8 public immutable ohmDecimals;
ERC20 public immutable reserve;
uint8 public immutable reserveDecimals;
uint32 public constant FACTOR_SCALE = 1e4;
```

[G-08] EXPRESSIONS THAT CANNOT BE OVERFLOWED CAN BE UNCHECKED (5 INSTANCES)

Deployment Gas Saved: 23 016

- src/Kernel.sol:299-300, 309-310

```
299         activePolicies.push(policy_);
300         getPolicyIndex[policy_] = activePolicies.length - 1; // @note cannot be over
...
309             moduleDependents[keycode].push(policy_);
310             getDependentIndex[keycode][policy_] = moduleDependents[keycode].length -
```

- src/modules/PRICE.sol:89, 144, 171

```
89          uint256 exponent = decimals + reserveEthDecimals - ohmEthDecimals; //@note ov
...
144         nextObsIndex = (nextObsIndex + 1) % numObs; //@note numObs can not be equal (
...
171             if (updatedAt < block.timestamp - uint256(observationFrequency)) // @note
```

[G-09] INCREMENT OPTIMIZATION (18 INSTANCES)

For a uint256 i variable, the following is true with the Optimizer enabled at 10k:

Increment:

i += 1 is the most expensive form
i++ costs 6 gas less than i += 1
++i costs 5 gas less than i++ (11 gas less than i += 1)

Decrement:

i -= 1 is the most expensive form
i— costs 11 gas less than i -= 1
--i costs 5 gas less than i— (16 gas less than i -= 1)

[G-09.1] Prefix increments are cheaper than postfix increments, especially when it's used in for-loops (3 instances).

Deployment Gas Saved: 400

- src/utils/KernelUtils.sol:49, 64

```
49            i++;
...
64            i++;
```

- src/policies/Operator.sol:488

NOTE: in case of 670 675 686 691 not applicable and gas will be lost

```
488             decimals++;
```

[G-09.2] `<x> = <x> + 1` even more efficient than pre increment.(18 instances)

Deployment Gas Saved: 14 217

- src/utils/KernelUtils.sol:49, 64

```
49              i++;
...
64              i++;
```

- src/policies/Operator.sol:488, 670, 675, 686, 691
- *

```
488              decimals++;
...
670                  _status.low.count++;
...
675                  _status.low.count--;
...
686                  _status.high.count++;
...
691                  _status.high.count--;
```

- src/Kernel.sol:313, 357, 369, 386, 404, 429

```
313                  ++i;
...
357                  ++i;
...
369                  ++j;
...
386                  ++i;
...
404                  ++i;
...
429                  ++i;
```

- src/modules/INSTR.sol:72

```
72                  ++i;
```

- src/modules/PRICE.sol:225

```
225                 ++i;
```

- src/policies/BondCallback.sol:163

```
163                 ++i;
```

- src/policies/Governance.sol:281

```
281                 ++step;
```

- src/policies/TreasuryCustodian.sol:62

```
62                 ++j;
```

[G-10] USE NAMED RETURNS FOR LOCAL VARIABLES WHERE IT IS POSSIBLE (3 INSTANCES)

Deployment Gas Saved: 5 400

- src/Kernel.sol:130-135

```
130    /// @notice Function to grab module address from a given keycode.
131    function getModuleAddress(Keycode keycode_) internal view returns (address) {
132        address moduleForKeycode = address(kernel.getModuleForKeycode(keycode_));
133        if (moduleForKeycode == address(0)) revert Policy_ModuleDoesNotExist(keycode_
134        return moduleForKeycode;
135    }
```

fix:

```
function getModuleAddress(Keycode keycode_) internal view returns (address moduleFor
    moduleForKeycode = address(kernel.getModuleForKeycode(keycode_));
    if (moduleForKeycode == address(0)) revert Policy_ModuleDoesNotExist(keycode_);
}
```

- src/modules/INSTR.sol:41-79

```
41    /// @notice Store a list of Instructions to be executed in the future.
42    function store(Instruction[] calldata instructions_) external permissioned returns
43        uint256 length = instructions_.length;
44        uint256 instructionsId = ++totalInstructions;
45
46        Instruction[] storage instructions = storedInstructions[instructionsId];
...
76        emit InstructionsStored(instructionsId);
77
78        return instructionsId;
79    }
```

- src/modules/PRICE.sol:153-180

```
153    /// @notice Get the current price of OHM in the Reserve asset from the price fee
154    function getCurrentPrice() public view returns (uint256) {
...
177        uint256 currentPrice = (ohmEthPrice * _scaleFactor) / reserveEthPrice;
178
179        return currentPrice;
180    }
```

[G-11] `X = X + Y` IS CHEAPER THAN `X += Y;` (6 INSTANCES)

Deployment Gas Saved: 5 000

Usually does not work with struct and mappings.

- src/modules/PRICE.sol:136, 138, 222

```
136             _movingAverage += (currentPrice - earliestPrice) / numObs;
...
138             _movingAverage -= (earliestPrice - currentPrice) / numObs;
...
222             total += startObservations_[i];
```

- src/modules/VOTES.sol:56, 58

```
56          balanceOf[from_] -= amount_;
...
58              balanceOf[to_] += amount_;
```

- src/policies/Heart.sol:103

```
103         lastBeat += frequency();
```

[G-12] DELETING A STRUCT IS CHEAPER THAN CREATING A NEW STRUCT WITH NULL VALUES. (1 INSTANCES)

Deployment Gas Saved: 4 207
Method Call Gas Saved: 40

- src/policies/Governance.sol:288

```
288         activeProposal = ActivatedProposal(0, 0);
```

fix:

```
        delete activeProposal;
```

[G-13] DON'T COMPARE BOOLEAN EXPRESSIONS TO BOOLEAN LITERALS (2 INSTANCES)

Deployment Gas Saved: 1 607

- src/policies/Governance.sol:223, 306

```
223         if (proposalHasBeenActivated[proposalId_] == true) {
...
306         if (tokenClaimsForProposal[proposalId_][msg.sender] == true) {
```

[G-14] REVERT OPERATOR SHOULD BE IN THE CODE AS EARLY AS REASONABLY POSSIBLE (3 INSTANCES)

Deployment Gas Saved: 200
Method Call Gas Saved: 1 559+

- src/modules/INSTR.sol:43-48

```
43        uint256 length = instructions_.length;
44        uint256 instructionsId = ++totalInstructions;
45
46        Instruction[] storage instructions = storedInstructions[instructionsId];
47
48        if (length == 0) revert INSTR_InstructionsCannotBeEmpty(); // @note after 43
```

- src/policies/Governance.sol:180-191, 241-249

```
180     function endorseProposal(uint256 proposalId_) external {
181         uint256 userVotes = VOTES.balanceOf(msg.sender); // @note put after revert
182
183         if (proposalId_ == 0) {
184             revert CannotEndorseNullProposal();
185         }
186
187         Instruction[] memory instructions = INSTR.getInstructions(proposalId_);
188         if (instructions.length == 0) {
189             revert CannotEndorseInvalidProposal();
190         }
191
```

```
        uint256 userVotes = VOTES.balanceOf(msg.sender); // @note put after revert

        if (activeProposal.proposalId == 0) {
            revert NoActiveProposalDetected();
        }

        if (userVotesForProposal[activeProposal.proposalId][msg.sender] > 0) {
            revert UserAlreadyVoted();
        }
```

[G-15] DUPLICATED REQUIRE()/REVERT() CHECKS SHOULD BE REFACTORED TO A MODIFIER OR FUNCTION

Method Call Gas Saved: 8 111

- src/modules/PRICE.sol

```
if (!initialized) revert Price_NotInitialized(); // @note 4 instances
```

0xLienid (Olympus) confirmed

Disclosures