SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: Olympus
Prepared by: Sherlock
Lead Security Expert: lllllll
Dates Audited: January 22 - January 25, 2024
Prepared on: February 6, 2024

Introduction

The Olympus protocol is a decentralized financial (DeFi) system that supports OHM, a treasury backed token on the Ethereum network.

Scope

Repository: OlympusDAO/bophades

Branch: governance-clean

Commit: 3c4098ef9b2870f4ebd912b15466780676ba7db8

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
4	0

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

IIIIIII
hals
cawfree
LTDingZhen
ck
fibonacci
haxatron
r0ck3tz
Kow

blutorque
emrekocak
pontifex

nobody2018
Bauer
cocacola

alexzoid
s1ce
Breeje

Issue M-1: Nobody can cast for any proposal

Source: https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance-judging/issues/37

Found by

Bauer, Breeje, alexzoid, blutorque, cawfree, cocacola, emrekocak, fibonacci, hals, nobody2018, pontifex, s1ce

Summary

[castVote](https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance/blob/main/bophades/src/external/governance/GovernorBravoDelegate.sol#L369)/[castVoteWithReason](https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance/blob/main/bophades/src/external/governance/GovernorBravoDelegate.sol#L385)/[castVoteBySig](https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance/blob/main/bophades/src/external/governance/GovernorBravoDelegate.sol#L403) are used to vote for the specified proposal. These functions internally call [castVoteInternal](https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance/blob/main/bophades/src/external/governance/GovernorBravoDelegate.sol#L433-L437) to perform voting logic. However, `castVoteInternal` can never be executed successfully.

Vulnerability Detail

```
File: bophades\src\external\governance\GovernorBravoDelegate.sol
433:     function castVoteInternal(
434:         address voter,
435:         uint256 proposalId,
436:         uint8 support
437:     ) internal returns (uint256) {
......
444:         // Get the user's votes at the start of the proposal and at the
 ↪  time of voting. Take the minimum.
445:         uint256 originalVotes = gohm.getPriorVotes(voter,
 ↪  proposal.startBlock);
446:->       uint256 currentVotes = gohm.getPriorVotes(voter, block.number);
447:         uint256 votes = currentVotes > originalVotes ? originalVotes :
 ↪  currentVotes;
......
462:     }
```

The second parameter of `gohm.getPriorVotes(voter, block.number)` can only a

number smaller than `block.number`. Please see the [code](https://etherscan.io/token/0x0ab87046fBb341D058F17CBC4c1133F25a20a52f#code#L703) deployed by gOHM on the mainnet:

```
function getPriorVotes(address account, uint256 blockNumber) external view
↪  returns (uint256) {
->      require(blockNumber < block.number, "gOHM::getPriorVotes: not yet
↪  determined");
......
    }
```

Therefore, L446 will always revert. Voting will not be possible.

Copy the coded POC below to one project from Foundry and run `forge test -vvv` to prove this issue.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.20;

import "forge-std/Test.sol";

interface CheatCodes {
    function prank(address) external;
    function createSelectFork(string calldata,uint256) external returns(uint256);
}

interface IGOHM {
    function getPriorVotes(address account, uint256 blockNumber) external view
↪  returns (uint256);
}

contract ContractTest is DSTest{
    address gOHM = 0x0ab87046fBb341D058F17CBC4c1133F25a20a52f;
    CheatCodes cheats = CheatCodes(0x7109709ECfa91a80626fF3989D68f67F5b1DD12D);

    function setUp() public {
        cheats.createSelectFork("https://rpc.ankr.com/eth", 19068280);
    }

    function testRevert() public {
        address user = address(0x12399543949349);
        cheats.prank(user);
        IGOHM(gOHM).getPriorVotes(address(0x1111111111), block.number);
    }

    function testOk() public {
        address user = address(0x12399543949349);
```

```
        cheats.prank(user);
        IGOHM(gOHM).getPriorVotes(address(0x1111111111), block.number - 1);
    }
}
/**output
[PASS] testOk() (gas: 13019)
[FAIL. Reason: revert: gOHM::getPriorVotes: not yet determined] testRevert()
↪  (gas: 10536)
Traces:
  [10536] ContractTest::testRevert()
    [0] VM::prank(0x0000000000000000000000012399543949349)
        ()
    [540] 0x0ab87046fBb341D058F17CBC4c1133F25a20a52f::getPriorVotes(0x000000000
↪  0000000000000000000001111111111, 19068280 [1.906e7]) [staticcall]
        revert: gOHM::getPriorVotes: not yet determined
     revert: gOHM::getPriorVotes: not yet determined

Test result: FAILED. 1 passed; 1 failed; 0 skipped; finished in 1.80s
**/
```

Impact

Nobody can cast for any proposal. Not being able to vote means the entire governance contract will be useless. Core functionality is broken.

Code Snippet

https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance/blob/main/bophades/src/external/governance/GovernorBravoDelegate.sol#L446

Tool used

Manual Review

Recommendation

Discussion

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

haxatron commented:

> Medium. It would be caught immediately on deployment and implementation is upgradeable. There can be no loss of funds which is requisite of a high.

IIIIIII000

Agree with haxatron that this is Medium, not High, based on Sherlock's rules

nevillehuang

Can agree, since this is purely a DoS, no malicious actions can be performed since no voting can be done anyways.

@Czar102 I am interested in hearing your opinion, but I will set medium for now, because governance protocols fund loss impact is not obvious but I initially rated it high because it quite literally breaks the whole protocol. I believe sherlock needs to cater to different types of protocols and not only associate rules to defi/financial losses (example protocols include: governance, on chain social media protocols etc..)

0xLienid

Fix: https://github.com/OlympusDAO/bophades/pull/293

IIIIIII000

The PR follows the suggested recommendation and correctly modifies the only place that solely `block.number` is used, changing it to `block.number - 1`. The only place not using this value is the call above it which uses `proposal.startBlock`. The `state()` when `startBlock` is equal to `block.number` is `ProposalState.Pending`, so this case will never cause problems, since there are checks of the state. The PR also modifies the mock gOHM contract to mirror the behavior that caused the bug.

s1ce

Escalate

This is a high. Voting is a core part of a governance protocol, and this bricks all voting functionality.

sherlock-admin2

> Escalate
>
> This is a high. Voting is a core part of a governance protocol, and this bricks all voting functionality.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

0xf1b0

Besides the fact that this issue breaks the core logic of the contract, *it won't be immediately detected upon deployment*, as previously mentioned as the reason for downgrading the severity. The voting process only becomes possible after a proposal has been made and time has elapsed. At this point, the issue will be raised, necessitating the deployment of an update. While the new version is being prepared, the proposal may expire, and a new one will have to be created. If the proposal includes some critical changes, this time delay can pose a serious problem.

lllllll000

Ignore this part since, while true, it appears to be confusing some: ~~The sponsor mentioned this test file as where to look for how things will be deployed. The first action is to propose and start a vote for assigning the whitelist guardian, and that will flag the issue before anything else.~~

Furthermore, the timelock needs to pull in order to become and admin with access to the treasury. Until that happens, the existing admin has the power to do anything, so there's no case where something critical can't be done. The 'pull' requirement for transferring the admin to the timelock is a requirement of the code, not of the test. The Sherlock rules also state that opportunity costs (e.g. delays in voting `for example, due to a loss of core functionality`) do not count as a loss of funds.

r0ck3tzx

The test file within `setUp()` function configures the environment for testing, not for the actual deployment. The deployment process can and probably will look different, so no assumptions should be made based on the test file. The mention just shows how the `whitelistGuardian` will be configured, and not at what time/stage it will be done.

The LSW creates hypotheticals about how the deployment process might look, and because of that, the issue would be caught early. Anyone who has ever deployed a protocol knows that the process is complex and often involves use of private mempools. Making assumptions about the deployment without having actual deployment scripts is dangerous and might lead to serious issues.

0xf1b0

Even though some proposals may be initiated at the time of deployment, it will take between 3 to 7 days before the issue becomes apparent, as voting will not be available until then.

nevillehuang

I agree with watsons here, but would have to respect the decision of @Czar102 and

his enforcement of sherlock rules. Governance protocols already have nuances of funds loss being not obvious, and the whole protocol revolves around voting as the core mechanism, if you cannot vote, you essentially lose the purpose of the whole governance.

0xf1b0

I've seen a lot of discussion regarding the rule of funds at risk. It seems that they never take into account the lost profits. A scenario where the core functionality of the system is broken could result in a loss of confidence in the protocol, causing users to be hesitant about investing their money due to the fear of such an issue recurring.

Czar102

From my understanding, due to the fact that the timelock needs to pull, the new governance contract needs to call it. And since it's completely broken, it will never pull the admin rights.

Hence, this is not a high severity impact of locking funds and rights in a governance, but a medium severity issue since the contract fails to work. Is it accurate? @lllllll000

lllllll000

Yes, that's correct

Czar102

Planning to reject the escalation and leave the issue as is.

0xf1b0

By the way, it will not be possible to update the contract, because a new implementation can only be set through the voting process, which does not work.

That's at least 2 out of 3:

- it won't be immediately detected upon deployment
- it's not upgradeable

lllllll000

it's being deployed fresh for this project, so it'll just be redeployed. The 2/3 stuff I think you're referring to is for new contests

Czar102

Result: Medium Has duplicates

sherlock-admin

Escalations have been resolved successfully!

Escalation status:

- s1ce: rejected

Issue M-2: High risk checks can be bypassed with extra `calldata` padding

Source: https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance-judging/issues/100

Found by

lllllll, Kow, cawfree, fibonacci, haxatron, r0ck3tz

Summary

Adding extra unused bytes to proposal calldata can trick the `_isHighRiskProposal()` function

Vulnerability Detail

The length checks on the transaction calldata of what falls into the 'high risk' proposal category is too strict, and incorrectly fails with extra padding. In solidity, any extra bytes of `calldata`, beyond what is required to satisfy the function arguments, are ignored, and have no effect on the operation of the function being called.

Impact

A proposal that should have been flagged as high risk, is not, and therefore can be passed with the easier, lower, quorum. This violates a critical invariant.

Code Snippet

Checks for calls to the kernel's `executeAction()` function, expect exactly the right number of bytes to satisfy the function arguments, and no more:

```
// File: src/external/governance/GovernorBravoDelegate.sol :
↪  GovernorBravoDelegate._isHighRiskProposal()   #1

631                     // Check if the action is making a core change to system
↪  via the kernel
632                     if (selector == Kernel.executeAction.selector) {
633                         uint8 action;
634                         address actionTarget;
635
636 @>                      if (bytes(signature).length == 0 && data.length ==
↪  0x44) {
```

```
637                             assembly {
638                                 action := mload(add(data, 0x24)) //
↪  accounting for length and selector in first 4 bytes
639                                 actionTarget := mload(add(data, 0x44))
640                             }
641 @>                      } else if (data.length == 0x40) {
642                             (action, actionTarget) = abi.decode(data, (uint8,
↪  address));
643                         } else {
644                             continue;
645:                         }
```

https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance/blob/main/bophades/src/external/governance/GovernorBravoDelegate.sol#L631-L645

this results in an easier quorum threshold:

```
// File: src/external/governance/GovernorBravoDelegate.sol :
↪  GovernorBravoDelegate.propose()   #2

168                 // Identify the quorum level to use
169 @>              if (_isHighRiskProposal(targets, signatures, calldatas)) {
170                     quorumVotes = getHighRiskQuorumVotes();
171                 } else {
172 @>                  quorumVotes = getQuorumVotes();
173:                }
```

https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance/blob/main/bophades/src/external/governance/GovernorBravoDelegate.sol#L168-L173

Tool used

Manual Review

Recommendation

Change length checks to be >=, rather than strict equality, since the function signature already specifies the number of arguments

PoC

The following test shows that extending the calldata by an empty byte still triggers a valid call to `executeAction()`, but is categorized as lower severity:

```
diff --git a/bophades/src/test/external/GovernorBravoDelegate.t.sol
↪  b/bophades/src/test/external/GovernorBravoDelegate.t.sol
index 778163c..bdb6ae2 100644
--- a/bophades/src/test/external/GovernorBravoDelegate.t.sol
+++ b/bophades/src/test/external/GovernorBravoDelegate.t.sol
@@ -386,6 +386,10 @@ contract GovernorBravoDelegateTest is Test {
         assertEq(quorum, 200_000e18);
     }

+    function executeAction(Actions action_, address target_) external {
+        console2.log("executed with extra calldata");
+    }
+
     function testCorrectness_proposeCapturesCorrectQuorum_highRisk() public {
         // Activate TRSRY
         vm.prank(address(timelock));
@@ -404,9 +408,12 @@ contract GovernorBravoDelegateTest is Test {
         calldatas[0] = abi.encodeWithSelector(
             kernel.executeAction.selector,
             Actions.ActivatePolicy,
-            address(custodian)
+            address(custodian),
+            ""
         );

+        address(this).call(calldatas[0]);
+
         vm.prank(alice);
         bytes memory data = address(governorBravoDelegator).functionCall(
             abi.encodeWithSignature(
```

Output:

```
% forge test --match-test testCorrectness_proposeCapturesCorrectQuorum_highRisk
↪  -vv
...
[FAIL. Reason: assertion failed]
↪  testCorrectness_proposeCapturesCorrectQuorum_highRisk() (gas: 568208)
Logs:
  executed with extra calldata
  Error: a == b not satisfied [uint]
    Expected: 300000000000000000000000
      Actual: 200000000000000000000000

Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 14.92ms
```

```
...
```

Discussion

0xLienid

Valid, will fix by reverting if the calldata doesn't match the right size since we know what the size must be for an `executeAction` call

0xLienid

Fix: https://github.com/OlympusDAO/bophades/pull/299

IIIIIII000

The PR introduces a new revert error, and reverts if the length is longer than expected, rather than allowing the code to continue if the calldata is longer than expected. Since the selector ensures that the right number of arguments is passed, there is no error in restricting possible future uses of extra calldata. The PR also adds a test.

Issue M-3: Post-proposal vote quorum/threshold checks use a stale total supply value

Source: https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance-judging/issues/102

Found by

lllllll, hals

Summary

The pessimistic vote casting approach stores its cutoffs based on the total supply during proposal creation, rather than looking up the current value for each check.

Vulnerability Detail

`gOHM token holders can delegate their voting rights either to themselves or to an address of their choice. Due to the elasticity in the gOHM supply, and unlike the original implementation of Governor Bravo, the Olympus governance system relies on dynamic thresholds based on the total gOHM supply. This mechanism sets specific thresholds for each proposal, based on the current supply at that time, ensuring that the requirements (in absolute gOHM terms) for proposing and executing proposals scale with the token supply.` https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance/tree/main/bophades/audit/2024-01_governance#olympus-governor-bravo-implementation

The above means that over time, due to dynamic minting and burning, the total supply will be different at different times, whereas the thresholds/quorums checked against are solely the ones set during proposal creation.

Impact

DoS of the voting system, preventing proposals from ever passing, under certain circumstances

Consider the case of a bug where there is some sort of runaway death spiral bug or attack in the dymamic burning of gOHM (e.g. opposite of Terra/Luna), and the only fix is to pass a proposal to disable the module(s) causing a problem where everyone is periodically having their tokens burn()-from-ed. At proposal creation there are sufficient votes to pass the threshold, but after the minimum 3-day waiting period, the total supply has been halved, and the original proposer no longer has a sufficient quorum to execute the proposal (or some malicious user decides to cancel it, and there is no user for which `isWhitelisted()` returns true).

No proposal can fix the issue, since no proposal will have enough votes to pass, by the time it's time to vote. Finally, once the total supply reaches low wei amounts, the treasury can be stolen by any remaining holders, due to loss of precision:

- `getProposalThresholdVotes()`: min threshold is 1_000, so if supply is <100, don't need any votes to pass anything
- `getQuorumVotes()`: quorum percent is hard-coded to 20_000 (20%), so if supply drops below 5, quorum is zero
- `getHighRiskQuorumVotes()`: high percent is hard-coded to 30_000 (30%), so if supply drops below 4, quorum is zero for high risk

Code Snippet

The quorum comes from the total supply...

```
// File: src/external/governance/GovernorBravoDelegate.sol :
↪  GovernorBravoDelegate.getHighRiskQuorumVotes()   #1

698         function getQuorumVotes() public view returns (uint256) {
699             return (gohm.totalSupply() * quorumPct) / 100_000;
700         }
...
706         function getHighRiskQuorumVotes() public view returns (uint256) {
707             return (gohm.totalSupply() * highRiskQuorum) / 100_000;
708:        }
```

https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance/blob/main/bophades/src/external/governance/GovernorBravoDelegate.sol#L696-L708

...and is set during propose(), and checked as-is against the eventual vote:

```
// File: src/external/governance/GovernorBravoDelegate.sol :
↪  GovernorBravoDelegate.getVoteOutcome()   #2

804             } else if (
805                 (proposal.forVotes * 100_000) / (proposal.forVotes +
↪  proposal.againstVotes) <
806 @>              approvalThresholdPct ||
807 @>              proposal.forVotes < proposal.quorumVotes
808             ) {
809                 return false;
810:             }
```

https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance/blob/main/bophades/src/external/governance/GovernorBravoDelegate.sol#L804-L810

Tool used

Manual Review

Recommendation

Always calculate the quorum and thresholds based on the current `gohm.totalSupply()` as is done in the OZ implementation, and consider making votes based on the fraction of total supply held, rather than a raw amount, since vote tallies are affected too

Discussion

0xLienid

If votes are locked in at a maximum of the value a voter had at the time the proposal started I don't think it makes sense to use the current `totalSupply` at proposal queueing to determine success/meeting quorum. you want it to be a comparable value to the votes values, hence lock it in at proposal creation

lllllll000

Since the votes are locked in at the proposal start, then shouldn't the quorum be based on the total supply at that starting block, in order to have a comparable value? Right now the code is consistent with the proposal time only, which may have a vastly different total supply. Shouldn't the code always be consistent with total supply of whichever block is being checked? The votes would still be at the time of the start of voting, but when determining whether a proposal should be queueable/executable/cancelable, if everyone's token counts and the total supply has be halved, there has been no change in who logically would be best positioned to vote, but since the code compares against a stored raw value rather than a ratio, the proposal can fail through no fault of the proposer. They would be able to propose a new vote but be unable to use their old proposal, even though the ownership percentage is the same as before.

nevillehuang

@0xLienid I think @lllllll000 highlights a valid scenario where this can cause a significant issue, and makes a good point as to why OZ implements quorums and thresholds computation that way.

However, I can also see how this is speculating on emergency situations etc.., but I think in the context of a governance, it is reasonable given it is where sensitive actions are performed. @Czar102 What do you think of this?

0xLienid

I just don't agree that you want the quorum to be subject to deviations in supply during the voting period. It allows user manipulation of the ease/difficulty required to pass a proposal.

> shouldn't the quorum be based on the total supply at that starting block, in order to have a comparable value?

yes, but that's impossible with gOHM

Frankly, I feel like if there is a critical bug in the core voting token then it's pretty expected that the governance system is also broken.

The only tenable option I guess is getting rid of the pessimistic voting.

0xLienid

@nevillehuang @lllllll000 Do you guys have additional thoughts on this? I'm trying to think about how severe it actually is, and if there's any path to fixing it other than using the live total supply which feels more or less similar to using votes not pinned to a block which is bad.

lllllll000

@0xLienid When you say it's impossible with gOHM, I believe you mean that once the starting block has passed, that there's no way to get the total supply from that prior block. If that's what you meant, in order to get the total supply at the starting block, you could require that the proposal creator actually trigger the start of voting (within some grace period) with another transaction at some point after the projected start block based on the delay, and have that operation update the stored quorums and start block at that point, assuming that the old quorums are still valid. In reality, the proposer controls the block during which the start occurs anyway, since the proposal block is under their control, and the delay is known.

As for the remainder of the issue, I'm not familiar with all that is planned, but I don't think it would have to be a bug in the core voting token itself - it could be a kernel module that has a role that allows it to mint/burn gOHM, given some algorithm with a time delay. Once things are decentralized, it's difficult to be able to predict that that won't happen. You could create a new gOHM that checkpoints the prior total supply, and migrate the old token, but yeah, that would be a big change, and would likely require larger changes than can be done for this contest.

nevillehuang

Actually @lllllll000 will there even ever be a situation where gOHM would be burned to literal 100, 5 and 4 given gOHM holds 18 decimals? I think on second look this is low severity, given the protocol can easily just implement a sanity check where they block any proposals creation/execution/queue and allow cancellation once totalSupply reaches this extreme small values.

I'm guessing your issue also points to the possible decrease in absolute quorums not just solely small amounts, but I think that example is not realistic and representative enough. Or am I missing a possible scenario where gOHM supply can reach literal small weis of value?

@0xLienid maybe a possible fix would be to make quorum percentages adjustable? This could open to front-running attacks though so I'll have to think through it more.

lllllll000

@nevillehuang the 100/5/4 scenario is the end point after which everything can get stolen. Prior to that, this bug outlines that they can't stop an ongoing attack because creating a proposal to do so would never pass quorums due to the bug. This bug essentially was an elaboration of the issue described in the duplicate #74, to show that it's an issue with the underlying mechanics, rather than a one-time total supply discrepancy

nevillehuang

@lllllll000 Yup that is my initial thoughts as well, sorry that I got confused by that scenario. I will likely maintain this issue as medium severity, and facilitate discussions between us and sponsor for a potential fix, since it seems to be non trivial.

0xLienid

Ok @lllllll000 @nevillehuang just talked with the other devs for a bit and here's what we came up with.

1. Separate out proposal activation to another function so we can snapshot total supply more accurately
2. Set a minimum total supply such that proposing/queuing/executing ends up in the hands of an admin (and block the standard behavior for end users) if we fall below that. If you think about a burn bug of this magnitude it's a critical implosion of the protocol and so it makes sense to not rely on full on chain governance

Thoughts?

lllllll000

By activation, you mean the triggering of the start of voting like I described above, or do you mean something else? The docs mention a veto guardian that will eventually be set to the zero address. If there is a new admin for this case, it won't be able to do the same sort of relinquishment without having the end result of the attack being a locked treasury (assuming there's no governance bypass to access funds some other way). If that's acceptable, I believe your two changes will solve the issue.

0xLienid

Yep, triggering of the start of voting.

0xrusowsky

- https://github.com/OlympusDAO/bophades/pull/303

@llllllll000 ready for review

RealLTDingZhen

I thought about this issue while the contest was going on and didn't submit it, because I thought it's a design choice——Due to the highly variable totalsupply of gOHM, the proposer may need far more tokens than its initial amount to ensure that the proposal is not canceled, and opponents can use fewer votes to reject the proposal by mint more gOHM. By the way, the solution given by LSR is flawed——Attackers can manipulate totalsupply to cancel the proposer's proposal through flashloan.

llllllll000

The PR implements part of the discussed changes. There is a new gOHM total supply threshold, under which only emergency votes are allowed. The level is correctly many orders of magnitude above the levels at which quorums have loss of precision. Calls to `propose()` are prevented when the total supply falls below the cutoff. A new `emergencyPropose()` is added, which can only be called during emergencies, by the veto guardian. It does not have any threshold requirements, or any limit to the number of outstanding proposals (both good), and does not update `latestProposalIds` (ok). Besides those differences, the code looks like `propose()`. For `propose()` the end block and quorum are no longer initially set, and require the user to call `activate()` to set them instead. The `activate()` function requires a non-emergency, pending state, for the start block to have been passed, and to not have been activated before. The proposal can be reentrantly re-activated in the same way #62, but there aren't any negative effects outside of extra events. There is no cut-off of when a proposal can be activated, which may allow proposals to stay dormant for years, until they're finally activated. `activate()` cannot be called during emergencies, so proposals created before an emergency will expire if things don't get resolved. This also means `emergencyPropose()` does not require activation or any actual votes - just that delays have been respected (seems to be intended behavior). The `execute()` function does not require any specific state during emergencies for the veto admin, but `timelock.executeTransaction()` prevents calling it multiple times. The `state()` function has a new state for `Emergency`, which applies to any proposal created by the veto admin via `emergencyPropose()`. Bug Med: the proposal.proposalThreshold isn't updated during activate(), which can be many years after the initial proposal. Bug Low: extra events if reentrantly called Bug Low: emergency proposals (more than one is allowed) created during one emergency, can be executed during a later emergency

0xLienid

Refix: https://github.com/OlympusDAO/bophades/pull/334

We added a activation grace period (which is checked in the `state` call) so that proposals cannot sit unactivated for months or years. We believe this sufficiently reduces the `proposalThreshold` concern. We chose not to update the `proposalThreshold` at the time of activation with this added check because it feels backwards from a governance perspective to brick someone's proposal after proposing if their balance has not changed.

We also shifted certain components of the `proposal` struct modification above the `_isHighRiskProposalCheck` to prevent reentrancy.

IIIIIIIOOO

The PR correctly addresses the Medium and one of the Lows from item 7 of 9 of the prior review, while leaving the remaining Low about emergency proposals being able to be used across emergencies. The extra events issue is fixed by moving up the state variable assignments to above the high risk proposal checks. The Medium is addressed by introducing a new state variable to GovernorBravoDelegateStorageV1, rather than to GovernorBravoDelegateStorageV2, for a grace period. There is no setter for the variable, so it must be set during the call to `initialize()` as the penultimate argument, which is properly done, or by creating a new implementation with a setter function. The `initialize()` call bounds the value within reasonable limits. The min amount is set to 17280, which is 2.4 days, and the max is set to 50400, which is exactly 7 days. The `state()` function is properly changed to convert `Pending` to `Expired`, after the activation grace period. The code reverts with `GovernorBravo_Vote_Closed()` when activation is attempted after the grace period. The PR adds tests for the grace period and for the reentrancy issue With the death spiral issue and the activation grace period issue resolved, the issue of the total supply changing has been mitigated to low.

Issue M-4: High-risk actions aren't all covered by the existing checks

Source: https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance-judging/issues/104

Found by

lllllll, LTDingZhen, cawfree, ck

Summary

Things such as changing the list of high risk operations, or migrating kernels are not counted as high risk, even though they are high-risk

Vulnerability Detail

High risk modules are checked against a mapping, but the changing of values within the mapping is not marked as high risk.

In addition, the `MigrateKernel` action is not protected, even though it can brick the protocol

Impact

Allows an attacker to brick the protocol with a low threshold, or to remove the high-risk modules from the list of high risk modules, resulting in a lower threshold
Violates invariant of high-risk actions needing to be behind a higher quorum

Code Snippet

`MigrateKernel` isn't considered high-risk, and neither are calls to _setModuleRiskLevel():

```
// File: src/external/governance/GovernorBravoDelegate.sol :
↪  GovernorBravoDelegate._isHighRiskProposal()   #1

647 @>                      // If the action is upgrading a module (1)
648                         if (action == 1) {
649                             // Check if the module has a high risk keycode
650                             if
↪  (isKeycodeHighRisk[Module(actionTarget).KEYCODE()]) return true;
651                         }
652 @>                      // If the action is installing (2) or deactivating
↪  (3) a policy, pull the list of dependencies
```

```
653                         else if (action == 2 || action == 3) {
654                             // Call `configureDependencies` on the policy
655                             Keycode[] memory dependencies =
    ↪ Policy(actionTarget)
656                                 .configureDependencies();
657
658                             // Iterate over dependencies and looks for high
    ↪ risk keycodes
659                             uint256 numDeps = dependencies.length;
660                             for (uint256 j; j < numDeps; j++) {
661                                 Keycode dep = dependencies[j];
662                                 if (isKeycodeHighRisk[dep]) return true;
663                             }
664:                         }
```

https://github.com/sherlock-audit/2024-01-olympus-on-chain-governance/blob/main/bophades/src/external/governance/GovernorBravoDelegate.sol#L647-L664

Tool used

Manual Review

Recommendation

Add those operations to the high risk category

Discussion

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

haxatron commented:

> Medium. Bypass of a non-critical security feature for MigrateKernel(). I would say setModuleRiskLevel() part doesn't count because it requires 2 proposals to succeed. Nice catch!

0xLienid

Fix: https://github.com/OlympusDAO/bophades/pull/298

IIIIIII000

The PR properly adds the operations in the recommendation to the list of what's considered high risk, as well as the recommendations from all of the duplicates. This is done by returning `true` for anything with a target of the timelock or delegator, or for any kernel migration or executor change. The PR also adds tests.

s1ce

Escalate

This is an informational issue. There are comments in the code which specifically describe the modules that the sponsors consider to be high risk , so would consider this to be a design decision.

For example, the following comments:

```
// If the action is upgrading a module (1)
```

```
// If the action is installing (2) or deactivating (3) a policy, pull the list of dependencies
```

sherlock-admin2

> Escalate
>
> This is an informational issue. There are comments in the code which specifically describe the modules that the sponsors consider to be high risk , so would consider this to be a design decision.
>
> For example, the following comments:
>
> ```
> // If the action is upgrading a module (1)
> ```
>
> ```
> // If the action is installing (2) or deactivating (3) a policy, pull the list of dependencies
> ```

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

nevillehuang

@lllllll000 any comments? I think this is not informational, because sensitive actions like this should consistently have appropriate quorums in place and not break the high risk invariant allowing for execution with lower votes than normal. This cannot be seen as a documentation and design decision error given an explicit fix has been made.

However, considering the veto mechanism, I can see where @s1ce is coming from.

lllllll000

I think the Sherlock team will need to decide what they want to do for this sort of case, since bricking the protocol is an unambiguously dangerous ability, and the purpose of the feature is to prevent that sort of thing (or else why not just rely on vetos for everything). The readme also says `The proposal can be vetoed at any`

`time (before execution) by the veto guardian. Initially, this role will belong to the DAO multisig. However, once the system matures, it could be set to the zero address.`, so at some point in the future, there will be nobody to veto anything.

0xLienid

Personally think this is a medium. These are definitionally high risk changes.

nevillehuang

> I think the Sherlock team will need to decide what they want to do for this sort of case, since bricking the protocol is an unambiguously dangerous ability, and the purpose of the feature is to prevent that sort of thing (or else why not just rely on vetos for everything). The readme also says `The proposal can be vetoed at any time (before execution) by the veto guardian. Initially, this role will belong to the DAO multisig. However, once the system matures, it could be set to the zero address.`, so at some point in the future, there will be nobody to veto anything.

Extremely good point, I think this should remain as medium severity.

Czar102

Agree with @nevillehuang, @0xLienid and @lllllll000. It seems managing policies is strictly safer than migrating the whole kernel.

Planning to reject the escalation and leave the issue as is.

Czar102

Result: Medium Has duplicates

sherlock-admin

Escalations have been resolved successfully!

Escalation status:

- s1ce: rejected

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.