CAIRO SECURITY CLAN

EKUBO PROTOCOL'S GOVERNANCE

SECURITY ASSESSMENT REPORT

MAY 2024

Prepared for

EKUBO PROTOCOL

Contents

1 About Cairo Security Clan

Cairo Security Clan is a leading force in the realm of blockchain security, dedicated to fortifying the foundations of the digital age. As pioneers in the field, we specialize in conducting meticulous smart contract security audits, ensuring the integrity and reliability of decentralized applications built on blockchain technology.

At Cairo Security Clan, we boast a multidisciplinary team of seasoned professionals proficient in blockchain security, cryptography, and software engineering. With a firm commitment to excellence, our experts delve into every aspect of the Web3 ecosystem, from foundational layer protocols to application-layer development. Our comprehensive suite of services encompasses smart contract audits, formal verification, and real-time monitoring, offering unparalleled protection against potential vulnerabilities.

Our team comprises industry veterans and scholars with extensive academic backgrounds and practical experience. Armed with advanced methodologies and cutting-edge tools, we scrutinize and analyze complex smart contracts with precision and rigor. Our track record speaks volumes, with a plethora of published research papers and citations, demonstrating our unwavering dedication to advancing the field of blockchain security.

At Cairo Security Clan, we prioritize collaboration and transparency, fostering meaningful partnerships with our clients. We believe in a customer-oriented approach, engaging stakeholders at every stage of the auditing process. By maintaining open lines of communication and soliciting client feedback, we ensure that our solutions are tailored to meet the unique needs and objectives of each project.

Beyond our core services, Cairo Security Clan is committed to driving innovation and shaping the future of blockchain technology. As active contributors to the ecosystem, we participate in the development of emerging technologies such as Starknet, leveraging our expertise to build robust infrastructure and tools. Through strategic guidance and support, we empower our partners to navigate the complexities of the blockchain landscape with confidence and clarity.

In summary, Cairo Security Clan stands at the forefront of blockchain security, blending technical prowess with a client-centric ethos to deliver unparalleled protection and peace of mind in an ever-evolving digital landscape. Join us in safeguarding the future of decentralized finance and digital assets with confidence and conviction.

2 Disclaimer

Disclaimer Limitations of this Audit:

This report is based solely on the materials and documentation provided by you to Cairo Security Clan for the specific purpose of conducting the security review outlined in the Summary of Audit and Scoped Files. The findings presented here may not be exhaustive and may not identify all potential vulnerabilities. Cairo Security Clan provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, occurs entirely at your own risk.

Inherent Risks of Blockchain Technology:

Blockchain technology remains in its developmental stage and is inherently susceptible to unknown risks and vulnerabilities. This review is specifically focused on the smart contract code and does not extend to the compiler layer, programming language elements beyond the reviewed code, or other potential security risks outside the code itself.

Report Purpose and Reliance:

This report should not be construed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. No third party should rely on this report for any purpose, including making investment or purchasing decisions.

Liability Disclaimer:

To the fullest extent permitted by law, Cairo Security Clan disclaims all liability associated with this report, its contents, and any related services and products arising from your use. This includes, but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services:

Cairo Security Clan does not warrant, endorse, guarantee, or assume responsibility for any products or services advertised by third parties within this report, nor for any open-source or third-party software, code, libraries, materials, or information linked to, referenced by, or accessible through this report, its content, and related services and products. This includes any hyperlinked websites, websites or applications appearing on advertisements, and Cairo Security Clan will not be responsible for monitoring any transactions between you and third-party providers. It is recommended that you exercise due diligence and caution when considering any third-party products or services, just as you would with any purchase or service through any medium.

Disclaimer of Advice:

FOR THE AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, ACCESS, AND/OR USE, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

3 Executive Summary

This document presents the security review performed by Cairo Security Clan on the Ekubo protocol.

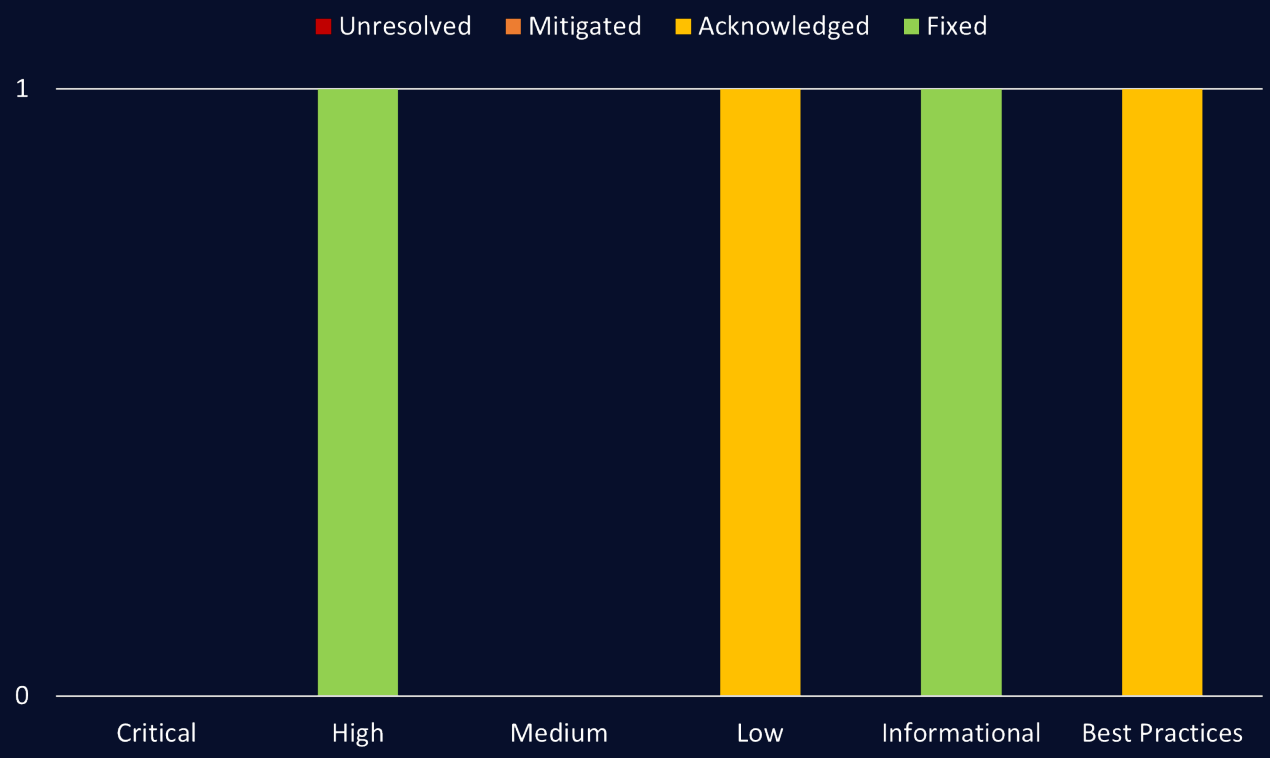
Ekubo Protocol is an automated market maker built for Starknet, with several unique features including concentrated liquidity, a singleton architecture, and extensions. It is designed to take full advantage of the Starknet architecture, with the goal of providing the best swapper execution and liquidity provider returns anywhere.Learn more from Ekubo docs.

The audit was performed using

- manual analysis of the codebase,
- automated analysis tools,
- simulation of the smart contract,
- analysis of edge test cases

4 points of attention, where 0 are classified as `Critical`, 1 are classified as `High`, 0 is classified as `Medium`, 1 are classified as `Low`, 1 are classified as `Info` and 1 are classified as `Best Practices`. The issues are summarized in Fig. 1.

This document is organized as follows. Section 1 About Cairo Security Clan. Section 2 Disclaimer. Section 3 Executive Summary. Section 4 Summary of Audit. Section 5 Risk Classification. Section 6 Issues by Severity Levels. Section 7 Test Evaluation.



(a) distribution of issues according to the severity and status

Fig 1: (a) Distribution of issues: Critical (0), **High** (1), **Medium** (0), **Low** (1), **Informational** (1), **Best Practices** (1). **(b) Distribution of status: Fixed** (2), **Acknowledged** (2), **Mitigated** (0), **Unresolved** (0)

4 Summary of Audit

Audit Type	Security Review
Cairo Version	2.6.3
Initial Report	29/04/2024
Response from Client	29/04/2024
Final Report	01/05/2024
Repository	EkuboProtocol/governance
Commit Hash	72544c4cc6ff65e1f0440a675622fddcea603438
Documentation	Website documentation
Test Suite Assessment	High

4.1 Scoped Files

	Governance Contracts
1	/src/airdrop.cairo
2	/src/airdrop_claim_check.cairo
3	/src/airdrop_test.cairo
4	/src/call_trait.cairo
5	/src/call_trait_test.cairo
6	/src/e2e_test.cairo
7	/src/execution_state.cairo
8	/src/execution_state_test.cairo
9	/src/call_trait.cairo
10	/src/factory.cairo
11	/src/factory_test.cairo
12	/src/governor.cairo
13	/src/governor_test.cairo
14	/src/lib.cairo
15	/src/staker.cairo
16	/src/staker_test.cairo
17	/src/timelock.cairo
18	/src/timelock_test.cairo
19	/src/interfaces/erc20.cairo
20	/src/test/test_token.cairo
21	/src/utils/exp2.cairo
22	/src/utils/u64_tuple_storage.cairo
23	/src/utils/u64_tuple_storage_test.cairo

4.2 Issues

	Findings	Severity	Update
1	Cancelled proposal can still be executed.	High	Fixed
2	Governor needs to create another proposal to cancel a queued call.	Low	Acknowledged
3	The `DelegatedSnapshot` field `delegated_cumulative` can overflow into `timestamp`.	Informational	Fixed
4	Unnecessary assertion.	Best Practices	Acknowledged

5 Risk Classification

The risk rating methodology used by Cairo Security Clan follows the principles established by the CVSS risk rating methodology. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely an attacker will uncover and exploit the finding. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to Motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Likelihood		
		High	**Medium**	**Low**
Impact	**High**	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Info/Best Practices

To address issues that do not fit a High/Medium/Low severity, Cairo Security Clan also uses three more finding severities: **Informational**, **Best Practices** and **Gas**

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to formally pass to the client;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

b) **Gas** findings are used when some piece of code uses more gas than it should be or have some functions that can be removed to save gas.

6 Issues by Severity Levels

6.1 High

6.1.1 Cancelled proposal can still be executed

File(s): /src/governor.cairo

Description: In the `governor` contract, each proposal's status is tracked by three timestamp values: `created`, `executed`, and `cancelled`. If a proposal is not executed, `executed` will be 0. If it is executed, this value will be set to the timestamp of execution. The same behavior applies to `created` and `cancelled`. These variables are checked in all core functions to ensure the correct state changes.

However, the `execute(...)` function does not verify if a proposal has been cancelled by checking `cancelled.is_zero()`. As a result, a cancelled proposal could still be executed if it received enough votes before being cancelled.

```
fn execute(ref self: ContractState, call: Call) -> Span<felt252> {
    let id = to_call_id(@call);

    let config = self.config.read();
    let mut proposal = self.proposals.read(id);

    assert(proposal.proposer.is_non_zero(), 'DOES_NOT_EXIST');
    assert(proposal.execution_state.executed.is_zero(), 'ALREADY_EXECUTED');
    ...
}
```

Recommendation(s): Consider adding a check to ensure cancelled proposals cannot be executed.

Status: Fixed

Update from client: Fixed in this commit

6.2 Low

6.2.1 Governor needs to create another proposal to cancel a queued call.

File(s): /src/timelock.cairo

Description: After a proposal is proposed, voted on, and executed, it is queued in the Timelock to be executed after the delay period. In the Timelock, only the owner can call the function `queue(...)`, and the owner is set to the Governor contract.

The function `cancel(...)` also has the same access control, meaning only the Governor contract can call it. Therefore, if we want to cancel a queued call, we need to go through the proposal-voting-execution process. This process could take longer than the delay of the Timelock, making it impossible to cancel a queued call.

```
fn queue(ref self: ContractState, calls: Span<Call>) -> felt252 {
    self.check_owner();
    ...
}

fn cancel(ref self: ContractState, id: felt252) {
    self.check_owner();
    ...
}
```

Recommendation(s): Consider allowing a different way to cancel a queued call in the Timelock, rather than forcing it to go through the voting process.

Status: Acknowledged

Update from client: The timelock is not necessarily owned by the governor. It can be used separately by any owner, including a regular account contract. If the governor wants to be able to cancel a proposal, the governor should set up the timelock delay to allow for time to execute a cancellation proposal. But the timelock's cancel function is not how governor proposals are meant to be canceled, so it's fine that you can't do it via proposal.

6.3 Informationals

6.3.1 The `DelegatedSnapshot` field `delegated_cumulative` can overflow into `timestamp`.

File(s): /src/staker.cairo

Description: The packing implementation for the `DelegatedSnapshot` struct compresses a `u64` timestamp and a `u256` cumulative delegated amount into a `felt252` field. The function is shown below:

```
fn pack(value: DelegatedSnapshot) -> felt252 {
    (value.delegated_cumulative
        + u256 { high: value.timestamp.into() * TWO_POW_64, low: 0 })
        .try_into()
        .unwrap()
}
fn unpack(value: felt252) -> DelegatedSnapshot {
    let (timestamp, delegated_cumulative) = DivRem::<
        u256
    >::div_rem(value.into(), u256 { low: 0, high: TWO_POW_64 }.into().try_into().unwrap());
    DelegatedSnapshot { timestamp: timestamp.low.try_into().unwrap(), delegated_cumulative }
}
```

The packed felt is created by multiplying the timestamp by `TWO_POW_64` to effectively act as a bitshift and placing the timestamp in the `high` section of the `u256`. This is then added to the cumulative delegated amount and converted to a `felt252`. Although unlikely, it is possible that the cumulative delegated amount can overflow into the bits reserved for the timestamp when these two `u256` variables are added together. As a result, the timestamp would have been increased incorrectly which can affect the function `find_delegated_cumulative` since it uses the timestamps to search through snapshots.

Recommendation(s): Consider splitting the stored data into two separate fields, or add a check to ensure that the cumulative delegated amount does not exceed `0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF` to prevent it from affecting the timestamp bits during addition.

Status: Fixed

Update from client: Added the assertion and some unit tests around max values in this commit.

6.4 Best Practices

6.4.1 Unnecessary assertion

File(s): /src/governor.cairo

Description: The function execute(...) contains an assertion that will always be true, asserting that the current block timestamp is non-zero. Given that the code already includes a comment stating this can only happen in testing, we recommend removing it for production.

```
fn execute(ref self: ContractState, call: Call) -> Span<felt252> {
  ...
  let timestamp_current = get_block_timestamp();
  // we cannot tell if a proposal is executed if it is executed at timestamp 0
  // this can only happen in testing, but it makes this method locally correct
  assert(timestamp_current.is_non_zero(), 'TIMESTAMP_ZERO');
  ...
}
```

Recommendation(s): Remove the unnecessary assertion.

Status: Acknowledged

Update from client: Per the comment, the assertion exists to make the method 'locally' correct. Meaning, that since we only use the bit timestamp != 0 as executed, an execution on block_timestamp = 0 should not be allowed so as to prevent execution from happening without being recorded.

7 Test Evaluation

7.1 Compilation Output

```
scarb build
   Compiling governance v0.1.0 (...\Scarb.toml)
    Finished release target(s) in 3 seconds
```

7.2 Tests Output

```
scarb test
     Running cairo-test governance
   Compiling test(governance_unittest) governance v0.1.0 (...\Scarb.toml)
    Finished release target(s) in 6 seconds
testing governance ...
running 106 tests
test governance::call_trait_test::test_execute_contract_not_deployed ... ok (gas usage est.: 75000)
test governance::call_trait_test::test_hash_address_entry_point_one ... ok (gas usage est.: 236180)
test governance::airdrop_test::test_selector ... ok (gas usage est.: 252290)
test governance::call_trait_test::test_hash_address_data_one ... ok (gas usage est.: 243480)
test governance::airdrop_test::test_hash ... ok (gas usage est.: 301820)
test governance::call_trait_test::test_execute_invalid_entry_point ... ok (gas usage est.: 243780)
test governance::call_trait_test::test_hash_address_data_one_two ... ok (gas usage est.: 250780)
test governance::airdrop_test::test_compute_pedersen_root_example_lt ... ok (gas usage est.: 279780)
test governance::call_trait_test::test_execute_invalid_call_data_too_short ... ok (gas usage est.: 253780)
test governance::airdrop_test::test_compute_pedersen_root_example_gt ... ok (gas usage est.: 279780)
test governance::governor_test::test_vote_invalid_proposal ... ok (gas usage est.: 894360)
test governance::airdrop_test::test_compute_pedersen_root_example_eq ... ok (gas usage est.: 279780)
test governance::airdrop_test::test_compute_root_of_group_empty ... ok (gas usage est.: 8940)
test governance::airdrop_test::test_compute_pedersen_root_empty ... ok (gas usage est.: 196040)
test governance::call_trait_test::test_execute_valid_call_data ... ok (gas usage est.: 465950)
test governance::airdrop_test::test_compute_root_of_group ... ok (gas usage est.: 923220)
test governance::airdrop_test::test_compute_pedersen_root_recursive ... ok (gas usage est.: 292040)
test governance::airdrop_test::test_compute_root_of_group_large ... ok (gas usage est.: 7130620)
test governance::airdrop_test::test_multiple_claims_from_generated_tree ... ok (gas usage est.: 2589206)
test governance::airdrop_test::test_compute_root_of_group_large_odd ... ok (gas usage est.: 7231890)
test governance::staker_test::test_get_delegated_at_fails_future_non_zero_ts ... ok (gas usage est.: 439680)
test governance::governor_test::test_proposer_can_cancel_and_propose_different ... ok (gas usage est.: 3938960)
test governance::governor_test::test_describe_proposal_fails_for_unknown_proposal ... ok (gas usage est.:
    3103450)
test governance::airdrop_test::test_claim_128_fails_if_not_id_aligned ... ok (gas usage est.: 618860)
test governance::governor_test::test_vote_before_voting_start_should_fail ... ok (gas usage est.: 3340440)
test governance::airdrop_test::test_claim_single_recipient ... ok (gas usage est.: 1668898)
test governance::governor_test::test_cancel_by_non_proposer_threshold_not_breached_should_fail ... ok (gas usage
    est.: 3690450)
test governance::governor_test::test_vote_after_cancel_proposal ... ok (gas usage est.: 3721580)
test governance::airdrop_test::test_claim_128_empty ... ok (gas usage est.: 557790)
test governance::airdrop_test::test_claim_128_single_recipient_tree ... ok (gas usage est.: 1785396)
test governance::airdrop_test::test_claim_128_too_many_claims ... ok (gas usage est.: 2435160)
test governance::governor_test::test_describe_proposal_fails_if_executed ... ok (gas usage est.: 6614450)
test governance::airdrop_test::test_claim_two_claims ... ok (gas usage est.: 2479876)
test governance::governor_test::test_execute_already_executed_should_fail ... ok (gas usage est.: 6266360)
test governance::staker_test::test_governance_token_delegated_snapshot_store_pack ... ok (gas usage est.: 600490)
test governance::staker_test::test_governance_token_delegated_snapshot_store_unpack ... ok (gas usage est.:
    708840)
test governance::governor_test::test_describe_proposal_fails_if_canceled ... ok (gas usage est.: 3530030)
test governance::airdrop_test::test_double_claim ... ok (gas usage est.: 1666866)
test governance::governor_test::test_vote_already_voted_should_fail ... ok (gas usage est.: 3974980)
test governance::e2e_test::test_create_proposal_that_fails ... ok (gas usage est.: 8429680)
test governance::governor_test::test_execute_before_voting_ends_should_fail ... ok (gas usage est.: 3334270)
test governance::governor_test::test_cancel_after_voting_end_should_fail ... ok (gas usage est.: 3308260)
test governance::staker_test::test_get_average_delegated_order_same ... ok (gas usage est.: 459310)
test governance::airdrop_test::test_claim_two_claims_via_claim_128 ... ok (gas usage est.: 2632698)
test governance::governor_test::test_propose_already_exists_should_fail ... ok (gas usage est.: 4828860)
test governance::governor_test::test_describe_proposal_fails_if_not_proposer ... ok (gas usage est.: 703520)
test governance::staker_test::test_get_average_delegated ... ok (gas usage est.: 5717890)
```

```
test governance::staker_test::test_get_average_delegated_order_backwards ... ok (gas usage est.: 459310)
test governance::airdrop_test::test_double_claim_128_single_recipient_tree ... ok (gas usage est.: 1859442)
test governance::airdrop_test::test_claim_three_claims_one_invalid_via_claim_128 ... ok (gas usage est.: 1314170)
test governance::staker_test::test_get_average_delegated_future_non_zero ... ok (gas usage est.: 460910)
test governance::governor_test::test_propose_below_threshold_should_fail ... ok (gas usage est.: 1242720)
test governance::staker_test::test_get_average_delegated_future ... ok (gas usage est.: 460310)
test governance::airdrop_test::test_invalid_proof_single_entry ... ok (gas usage est.: 928888)
test governance::factory_test::test_deploy ... ok (gas usage est.: 2169260)
test governance::governor_test::test_vote_no_staking_after_period_starts ... ok (gas usage est.: 5092740)
test governance::governor_test::test_to_call_id ... ok (gas usage est.: 472520)
test governance::staker_test::test_approve_sets_allowance ... ok (gas usage est.: 568140)
test governance::governor_test::test_vote_already_voted_different_vote_should_fail ... ok (gas usage est.:
    3974980)
test governance::governor_test::test_execute_quorum_not_met_should_fail ... ok (gas usage est.: 3336450)
test governance::airdrop_test::test_claim_from_end_of_tree ... ok (gas usage est.: 2269996)
test governance::staker_test::test_transfer_delegates_moved ... ok (gas usage est.: 2652510)
test governance::airdrop_test::test_invalid_proof_fake_entry ... ok (gas usage est.: 914928)
test governance::governor_test::test_setup ... ok (gas usage est.: 1591590)
test governance::staker_test::test_get_delegated_cumulative_fails_future_non_zero_ts ... ok (gas usage est.:
    402830)
test governance::airdrop_test::test_claim_from_end_of_tree_large ... ok (gas usage est.: 2356576)
test governance::staker_test::test_get_delegated_cumulative ... ok (gas usage est.: 2237770)
test governance::staker_test::test_delegate_count_lags ... ok (gas usage est.: 3135930)
test governance::staker_test::test_get_delegated_at_fails_future ... ok (gas usage est.: 439080)
test governance::governor_test::test_execute_valid_proposal ... ok (gas usage est.: 7124850)
test governance::governor_test::test_vote_after_voting_period_should_fail ... ok (gas usage est.: 3340540)
test governance::staker_test::test_get_delegated_cumulative_fails_future ... ok (gas usage est.: 402230)
test governance::governor_test::test_vote_yes ... ok (gas usage est.: 5564630)
test governance::governor_test::test_anyone_can_vote ... ok (gas usage est.: 3847400)
test governance::governor_test::test_execute_no_majority_should_fail ... ok (gas usage est.: 6170110)
test governance::airdrop_test::test_claim_from_end_of_tree_middle_of_bitmap ... ok (gas usage est.: 2356576)
test governance::timelock_test::test_deploy ... ok (gas usage est.: 339060)
test governance::governor_test::test_propose ... ok (gas usage est.: 3815380)
test governance::staker_test::test_delegate_undelegate ... ok (gas usage est.: 5837880)
test governance::staker_test::stake_withdraw::test_fails_allowance_large ... ok (gas usage est.: 921060)
test governance::airdrop_test::test_double_claim_after_other_claim ... ok (gas usage est.: 4209594)
test governance::airdrop_test::test_double_claim_from_generated_tree ... ok (gas usage est.: 2269996)
test governance::governor_test::test_describe_proposal_successful ... ok (gas usage est.: 4472670)
test governance::governor_test::test_cancel_by_proposer ... ok (gas usage est.: 4688380)
test governance::timelock_test::test_queue_execute ... ok (gas usage est.: 1891910)
test governance::staker_test::stake_withdraw::test_fails_insufficient_balance ... ok (gas usage est.: 1245930)
test governance::airdrop_test::test_claim_before_funded ... ok (gas usage est.: 1042718)
test governance::governor_test::test_propose_has_active_proposal ... ok (gas usage est.: 3096780)
test governance::staker_test::stake_withdraw::test_takes_approved_token ... ok (gas usage est.: 2590080)
test governance::airdrop_test::refundable::test_not_refundable_reverts_after_time ... ok (gas usage est.: 535590)
test governance::call_trait_test::test_hash_empty_different_state ... ok (gas usage est.: 254050)
test governance::call_trait_test::test_hash_empty ... ok (gas usage est.: 236180)
test governance::timelock_test::test_queue_execute_twice ... ok (gas usage est.: 1958700)
test governance::call_trait_test::test_hash_address_one ... ok (gas usage est.: 236180)
test governance::timelock_test::test_queue_cancel ... ok (gas usage est.: 1469920)
test governance::utils::u64_tuple_storage_test::test_two_tuple_storage_forward_back ... ok (gas usage est.:
    1520650)
test governance::timelock_test::test_queue_executed_too_late ... ok (gas usage est.: 1419190)
test governance::airdrop_test::refundable::test_refundable_transfers_token_at_refund_timestamp ... ok (gas usage
    est.: 4042140)
test governance::timelock_test::test_queue_executed_too_early ... ok (gas usage est.: 1419660)
test governance::governor_test::test_proposer_cannot_cancel_and_re_propose ... ok (gas usage est.: 3469950)
test governance::airdrop_test::refundable::test_not_refundable_reverts ... ok (gas usage est.: 534990)
test governance::airdrop_test::refundable::test_refundable_reverts_before_timestamp ... ok (gas usage est.:
    832670)
test governance::governor_test::test_verify_votes_are_counted_over_voting_weight_smoothing_duration_from_start
    ... ok (gas usage est.: 7635580)
test governance::governor_test::test_cancel_by_non_proposer ... ok (gas usage est.: 5843130)
test governance::airdrop_test::test_claim_128_double_claim ... ok (gas usage est.: 135114234)
test governance::airdrop_test::test_claim_128_large_tree ... ok (gas usage est.: 131385054)
test result: ok. 106 passed; 0 failed; 0 ignored; 0 filtered out;
```