SPEARBIT

Berachain Governance Security Review

Auditors

Xmxanuel, Lead Security Researcher

Tnch, Security Researcher

Report prepared by: Lucas Goiriz

October 24, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Berachain is an EVM-identical L1 turning liquidity into security powered by Proof Of Liquidity.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of contracts-monorepo according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 3 days days in total, Berachain engaged with Spearbit to review the contracts-monorepo protocol. In this period of time a total of **8** issues were found.

Summary

Project Name	Berachain
Repository	contracts-monorepo
Commit	abbbf9...8f0d
Type of Project	L1, Governance
Audit Timeline	Oct 1 to Oct 3
Two week fix period	Oct 18

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	3	3	0
Low Risk	1	1	0
Gas Optimizations	0	0	0
Informational	4	1	3
Total	**8**	**5**	**3**

5 Findings

5.1 Medium Risk

5.1.1 Frontrunners can corrupt the initialization of the `TimeLock` and `BerachainGovernance` contracts

Severity: Medium Risk

Context: DeployGovernance.s.sol#L49, DeployGovernance.s.sol#L63

Description: The deployment process implemented in `DeployGovernance.s.sol` begins with two transactions (see DeployGovernance.s.sol#L39-L44) that deploy the corresponding implementation and proxy contracts for `BerachainGovernance` and `TimeLock`.

However, the initialization of the proxies' state is not done in those deployment transactions. Instead, it's done afterwards, in two individual transactions that call the contracts' `initialize` function (see DeployGovernance.s.sol#L49 and DeployGovernance.s.sol#L63).

This means that, after the deployment transactions and before the initialization transactions, the proxies are temporarily left in an uninitialized state. Then, anyone can frontrun the initialization transactions and initialize the proxies of `TimeLock` and `BerachainGovernance` with malicious parameters to corrupt the deployment process.

Recommendation: The deployment and initialization operations should be done atomically, so that proxies are never left in an uninitialized state. One possible implementation could involve moving the deployment process to a single smart contract that can deploy the contracts and initialize the proxies in a single transaction.

Berachain: Fixed in commit 60195c68.

Spearbit: Resolved.

5.1.2 `proposer` cannot `cancel` a proposal with a `GOV_VOTING_DELAY=0` after propose `block.timstamp + 1`

Severity: Medium Risk

Context: DeployGovernance.s.sol#L19

Description: The OZ governance contracts include a parameter called `votingDelay`, which specifies a time delay between the `block.timestamp` of the proposal's creation and the start of the voting process. The `votingDelay` determines the timestamp for the ERC20 balance snapshot. Before voting begins, the state is referred to as `Pending` in the OZ governance contracts.

In the default setup, proposals can only be canceled while in the `Pending` state.

- **GovernanceUpgradable.cancel**

```
// public cancel restrictions (on top of existing _cancel restrictions).
_validateStateBitmap(proposalId, _encodeStateBitmap(ProposalState.Pending));
```

However, in the Berachain deployment script, the `GOV_VOTING_DELAY` is set to 0. The internal clock in the governor contract is based on blocks and the `GOV_VOTING_DELAY=0` is converted to `1 block` because the timeToBlocks helper function (See: DeployGovernance.s.sol#L97) has a fallback for zero blocks:

```
// Fallback for safety:
if (blocks == 0) {
    blocks = 1;
}
```

A proposal can only be canceled at the `propose` timestamp + 1. As a result, it is nearly impossible to cancel a proposal after its creation for the user.

Recommendation: The need to cancel a proposal can arise quickly, for example through a configuration error. If it is not possible to cancel the proposal, user might accidentally vote for an incorrect one.

Consider allowing proposals to be canceled even when GOV_VOTING_DELAY=0. From a technical standpoint, a proposal could be canceled in any state except ProposalState.Canceled, ProposalState.Expired, and ProposalState.Executed.

Berachain: Fixed in commit 424f533d.

Spearbit: Resolved.

5.1.3 BerachainGovernance.updateTimelock can break the upgradability with _authorizeUpgrade

Severity: Medium Risk

Context: BerachainGovernance.sol#L50

Description: BerachainGovernance inherits from OwnableUpgradeable and GovernorTimelockControlUpgradeable, both initialized with the same timelock contract.

```
__GovernorTimelockControl_init(_timelock);
__Ownable_init(address(_timelock));
```

The timelock contract executes successful governance proposals after a delay.

The Ownable.onlyOwner modifier is used exclusively for _authorizeUpgrade, which manages upgrade authorization.

However, when the timelock is updated via the updateTimelock function, the contract's owner is not updated, causing the upgradability feature to break since ownership still points to the old timelock. This creates a situation where calling transferOwnership becomes impossible, as only the old timelock can authorize it.

The only possible workaround would be to reset the timelock to the old timelock contract, assuming the old timelock still has the required roles for the governance contract.

Recommendation: To avoid this issue, the transferOwnership call must occur before the timelock update. Alternatively, remove Ownable inheritance and update _authorizeUpgrade to only use the timelock stored in GovernorTimelockControl

```
function _authorizeUpgrade(address newImplementation) internal override {
    // _executor() returns the stored timelock from GovernorTimelockControl
    require(msg.sender == _executor());
}
```

Berachain: Fixed in commit 60195c68.

Spearbit: Resolved.

5.2 Low Risk

5.2.1 Weak post-deployment verifications in governance deployment script

Severity: Low Risk

Context: DeployGovernance.s.sol

Description: The `DeployGovernance.s.sol` script is intended to deploy and configure the `TimeLock` and `BerachainGovernance` contracts. Although the script does include some basic logging operations and one verification (see DeployGovernance.s.sol#L67-L70), it can benefit from more thorough verifications and logging of relevant post-deployment conditions of the governance system.

Recommendation: Consider implementing post-deployment verifications to check that:

- All settings declared at the start of the script are indeed set in the corresponding contracts.
- All proxies are initialized.
- The deployer account (`msg.sender` in the current version) has lost the `DEFAULT_ADMIN_ROLE` role in the `TimeLock` contract.
- The `TimeLock` contract is still self-administered, holding the `DEFAULT_ADMIN_ROLE` role.
- The proposer and executor accounts are set as expected in the `TimeLock` contract.
- If non-zero, then the `GOV_GUARDIAN` account has the `CANCELLER_ROLE` role assigned in the `TimeLock` contract.
- The `GovernanceBerachain` contract is configured to use the expected voting token, the deployed `TimeLock` contract, and is initialized with intended parameters (see DeployGovernance.s.sol#L56-L61).

We also suggest logging the addresses of all deployed proxy and implementation contracts.

Consider implementing the mentioned post-deployment verifications and logging operations to ensure these sensitive governance contracts are left in the expected state with the correct settings.

Do note that we're reporting another issue in the existing deployment script which may require moving the deployment code to a single smart contract. In such case, the post-deployment verifications could be moved to the new deployment contract as well.

Berachain: Fixed in commit 60195c69.

Spearbit: Resolved.

5.3 Informational

5.3.1 Use specific `salt` values for `create2` instead of `0`

Severity: Informational

Context: DeployGovernance.s.sol#L33

Description: In the depoy script the `salt` for the `BerachainGovernance` and `Timelock` contract used with `create2` is both zero.

```
/// @notice The CREATE2 salts to use for address consistency
uint256 internal constant GOV_CREATE2_NONCE = 0;
uint256 internal constant TIMELOCK_CREATE2_NONCE = 0;
```

Recommendation: Use a domain-specific salt like `berachain.governance` for the deployment. This allows a better management of different versions or potentially some test deployments on the same chain.

The address consistency on multiple different chains would not be impacted by more specific `salt` names if the same `CREATE2` factory contract (under the same address) is used.

Berachain: The zero values are currently placeholders because the final convention has not been decided.

Spearbit: Acknowledged.

5.3.2 Guardian `cancel` role on the `timelock` contract is a single point of failure

Severity: Informational

Context: DeployGovernance.s.sol#L51

Description: The current Berachain governance has a guardian role, which has the power to `cancel` any proposal from execution when it is queued in the timelock contract. Under the current configuration, the timelock contract can only execute a proposal after 2 days.

Theoretically, if the guardian keys were compromised the guardian could block any proposal from execution. Including the removal or adding a new guardian address.

Recommendation: Consider the additional risks associated with such a powerful role. In case the guardian role is required should be a multi-sig with a minimum signers threshold to prevent potential attacks.

Berachain: We are aware of the power of this role. The guardian is expected to be a 3-out-of-5 multi-sig with competent and trusted actors.

Spearbit: Acknowledged.

5.3.3 Skipping the remaining voting period when a `quorum` is reached with a current `yes` outcome changes voting dynamics

Severity: Informational

Context: BerachainGovernance.sol#L83

Description: The `BerachainGovernance` contract modifies the OpenZeppelin (OZ) governance state function to immediately skip the remaining voting period once the quorum is reached, and if the current outcome would result in a "yes" vote.

This provides the advantage of immediately queueing a proposal for execution once the quorum is met. However, this also implies that a single token holder, or a group of token holders, controlling 10% of the tokens could potentially propose and execute any proposal. In this scenario, the guardian within the timelock contract would serve as the last line of defense to cancel such transactions. Token holders could propose and vote as early as `block.timestamp + 1` to win the vote.

In standard OpenZeppelin implementations, potential "no" voters still have time to cast their votes until the voting period ends. OZ even offers a `GovernorPreventLateQuorum` extension to extend the voting period after the quorum is reached, preventing large token holders from voting at the last possible blocks to influence the outcome.

In the `BerachainGovernance` no-voters are encouraged to vote as early as possible to prevent successful `yes` outcome if a quorum is reached before the voting period ends. This is not the case for the `yes` voters.

Recommendation: The differing voting behaviors of `yes` and `no` voters should be clearly documented. Additionally, it is important to reassess whether the potential benefits of early proposal execution outweigh the increased risk.

Berachain: The accelerated proposal execution will remain in place for now due to non-technical considerations. The default behavior will be restored in a future upgrade when the guardianship will also be removed.

Spearbit: Acknowledged.

5.3.4 Missing calls to `UUPSUpgradeable.__UUPSUpgradeable_init` in initialization functions

Severity: Informational

Context: BerachainGovernance.sol#L55-L58, TimeLock.sol#L27

Description: The `initialize` functions of the `BerachainGovernance` and `TimeLock` contracts intend to execute their parents' initialization functions to follow the initialization chain. This is done even in cases when the parent's initialization function is empty, such as in the calls to `__GovernorCountingSimple_init` and `__GovernorStorage_-init`.

However, the `initialize` functions in both contracts are missing the call to the `__UUPSUpgradeable_init` function of the `UUPSUpgradeable` contract.

Recommendation: To consistently respect the initialization chain of contracts up the inheritance chain, consider adding the missing calls to `__UUPSUpgradeable_init` both in `BerachainGovernance.initialize` and `Time-Lock.initialize`.

Berachain: Fixed in commit 60195c68.

Spearbit: Resolved.