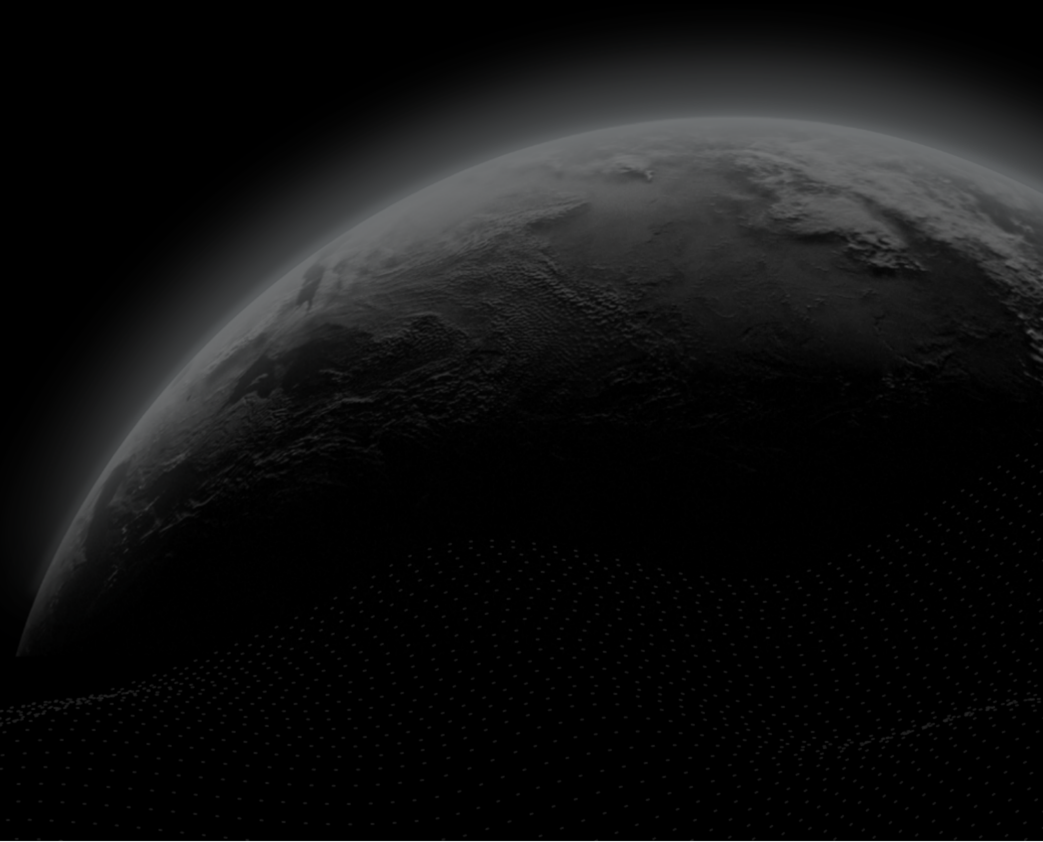
CERTIK

Security Assessment

JokeRace

CertiK Assessed on Sept 25th, 2023



CertiK Assessed on Sept 25th, 2023

JokeRace

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Governance	Ethereum (ETH)	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 09/25/2023	N/A

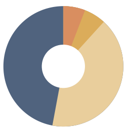
CODEBASE
jokerace
View All in Codebase Page

COMMITS
scope
remediation
View All in Codebase Page

Highlighted Centralization Risks

Privileged role can remove users' tokens

Vulnerability Summary



17	15	0	0	2	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

🟥 0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧 1	Major	1 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟨 1	Medium	1 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨 7	Minor	6 Resolved, 1 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦 8	Informational	8 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | JOKERACE

CODEBASE | JOKERACE

Repository

jokerace

Commit

scope

remediation

AUDIT SCOPE | JOKERACE

9 files audited ● 4 files with Acknowledged findings ● 3 files with Resolved findings ● 2 files without findings

ID	File	SHA256 Checksum
● GOV	packages/forge/src/governance/Governor.sol	5b2dbe52239a7a8fbe107a026db03ef5e23aae64c8af93314fb3d6349fa3c27a
● GSB	packages/forge/src/governance/extensions/GovernorSorting.sol	92d1e4a328fe7750552bda153a46742ea623124ec4c1e3dd4653f32a3a640a74
● GCS	packages/forge/src/governance/extensions/GovernorCountingSimple.sol	efbbc912353fcca268f15cc461ec20d05f085193caffd897c934f7b2f1863a45
● RMB	packages/forge/src/modules/RewardsModule.sol	4c9ba2d5c141b497c13431ffd0126f6ab8d42e516032b07cde1a7572538abd29
● GMV	packages/forge/src/governance/GovernorMerkleVotes.sol	a8a9d5c19a2abfc6bc02399cd101ff38fe332634e1869ef92b2783f5c4c13844
● IGB	packages/forge/src/governance/IGovernor.sol	2a011a8484818a71b056aa277a0399a84dda3d45a8bb7136fd9e3d7fd0579df1
● GMR	packages/forge/src/governance/extensions/GovernorModuleRegistry.sol	1a83c632018c7c3af20d75b693a00e3960e040db1dc66b1e6573cd7e9e217aff
● CON	packages/forge/src/Contest.sol	968f229068cdbe03278a39429fbc945b1eed05e847f45ba7bde66854286696e8
● GSU	packages/forge/src/governance/extensions/GovernorSettings.sol	1d346c070f258a5a65fe1e1e7b250dcbd09d8dd057ef3546bc916f8a74056007

APPROACH & METHODS | JOKERACE

This report has been prepared for JokeRace to discover issues and vulnerabilities in the source code of the JokeRace project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

SYSTEM OVERVIEW | JOKERACE

The contracts in scope realize an on-chain service to vote on a list of proposals within a defined timeframe. When a voting contest ends, a dedicated contract can handle the distribution of amounts of the native chain currency and ERC20 tokens to the proposers of the contest according to the final ranking of the proposals. The shares of each ranking position can be customized, but they can not be updated once defined.

Vote results are public and there is not any hiding mechanism of ballots, so the result can be tracked during the vote session. Such fact should be carefully taken into account since the contract creator can cancel the contest at any time.

The contracts' creator grant privileges to users allowing them to submit new proposals and/or vote during the contest period. Users' eligibility is verified through two Merkle trees, one for checking the submission privilege and one for the voting one.

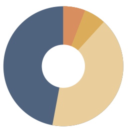
On the code organization side, the business logic is implemented across multiple contracts, each one incrementally adding new features and allowing for further extensions of the currently implemented logic. The top level contract, `Contest` , extends the other `Governor` -related contracts to leverage their functionalities. Rewards distribution is managed by the `RewardModule` contract which obtains contest's data by calling the `Contest` contract.

REVIEW NOTES | JOKERACE

The contracts in scope are well documented. All contracts describe their main logic while functions briefly specify their actions and document the required parameters.

Tests are included in the repository, they cover the successful behaviors and the main error/revert conditions. Further unit and integration tests should be implemented to ensure the correct code behavior in all revert and edge cases.

FINDINGS | JOKERACE



17	0	1	1	7	8
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for JokeRace. Through this audit, we have uncovered 17 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
GOV-03	**Centralization Related Risks**	**Centralization**	**Major**	● **Acknowledged**
RMB-04	Unable To Access Tied Proposals	Logical Issue	Medium	● Resolved
GCS-01	Incomplete Detection Of First Vote	Logical Issue	Minor	● Resolved
GOV-04	Duplicated Check In `validateProposalData`	Logical Issue	Minor	● Resolved
GOV-05	Locked Blockchain Native Tokens	Inconsistency	Minor	● Resolved
GSB-01	Potential Out-Of-Gas Exception	Logical Issue	Minor	● Acknowledged
GSB-04	Unsafe Integer Cast	Incorrect Calculation	Minor	● Resolved
RMB-02	Potential Reentrancy Attack (Out-Of-Order Events)	Concurrency	Minor	● Resolved
RMB-03	Potential Divide By Zero	Logical Issue	Minor	● Resolved
GMV-01	Missing Parameter Comment	Inconsistency	Informational	● Resolved
GOV-06	Unused Constant	Logical Issue	Informational	● Resolved

ID	Title	Category	Severity	Status
GOV-07	Creator Can Cancel An Active Contest	Logical Issue	Informational	● Resolved
GOV-08	Empty Parameter In `_castVote`	Inconsistency	Informational	● Resolved
GSB-05	Comparison To Boolean Constant	Coding Style	Informational	● Resolved
IGB-01	Hardcoded Enum Count	Coding Style	Informational	● Resolved
SRC-03	Missing Error Messages	Coding Style	Informational	● Resolved
SRC-04	Ranking Calculation On Tied Proposals	Logical Issue	Informational	● Resolved

GOV-03 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	● **Major**	**packages/forge/src/governance/Governor.sol: 276**	● **Acknowledged**

Description

The role `creator` has authority over the following functions:

- `Governor.deleteProposals`
- `Governor.cancel`
- `GovernorModuleRegistry.setOfficialRewardsModule`
- `RewardsModule.withdrawRewards`
- `RewardsModule.withdrawRewards(IERC20)`

Any compromise to the `creator` account may allow the hacker to take advantage of this authority and arbitrarily, respectively:

- delete proposals in active contests;
- move the contest in the cancelled state;
- set a malicious rewards module;
- drain the native currency balance of the `RewardsModule` contract;
- drain any ERC20 balance of the `RewardsModule` contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[`CertiK`]: The team acknowledged the finding and decided to remain unchanged as the privileges of the `creator` account are part of the intended design.

RMB-04 | UNABLE TO ACCESS TIED PROPOSALS

Category	Severity	Location	Status
Logical Issue	● Medium	packages/forge/src/modules/RewardsModule.sol: 277~279	● Resolved

Description

The code at the pointed lines accesses the proposal referred by a certain ranking. However, in the case two or many proposals have the same amount of votes, only one of them is accessed in reference to a certain ranking.

Additionally, there is no apparent mechanism to handle the amount distribution in the case of tied proposals, since only one of them is considered.

Recommendation

We recommend (1) to clarify which is the supposed behavior of the `RewardModule` contract when amount of cryptocurrency are distributed to tied proposals and (2) to implement the respective mechanism in the contract code, as the information about proposals referring to the same ranking position is lost in the `GovernorSorting` contract, above all for draws involving more than 2 proposals.

Alleviation

[`CertiK`]: The team clarified that the intended behavior is to pay rewards out to the `creator` account in case of tied proposal. Since this is the expected behavior, no action is required on the codebase.

GCS-01 | INCOMPLETE DETECTION OF FIRST VOTE

Category	Severity	Location	Status
Logical Issue	● Minor	packages/forge/src/governance/extensions/GovernorCountingSimple.sol: 101	● Resolved

Description

The validation at the pointed line checks if the account is voting for the first time by checking its `forVotes` field in the `ProposalVote` of the provided proposal ID.

However, in case `downvotingAllowed` is `1`, then an account can also vote with the `VoteType.Against` case, which sets the `againstVotes` in the `ProposalVote` struct. In such a case, if the account votes again using its left vote weight, it would be flagged again as a new voter and added again in the `addressVoted` array in Line 115.

This would cause a state inconsistency because of the double account in the array and would use more on-chain storage than required.

Recommendation

We recommend checking if an account already voted accounting for both the `For` and `Against` cases. An alternative solution would also be to convert `addressVoted` from an array to a `mapping`.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit 22cba1098ad6c32baf7b5f69c8d56013995cc633

GOV-04 | DUPLICATED CHECK IN `validateProposalData`

Category	Severity	Location	Status
Logical Issue	● Minor	packages/forge/src/governance/Governor.sol: 250	● Resolved

Description

The check at the pointed line verifies that the new proposal contains at least 1 signer. However, the same check is already performed in line 242.

At the same time a validation that the signers' threshold is less than the amount of signers is missing.

Recommendation

We recommend reviewing the pointed line of code, remove the duplicated check and add the suggested one to ensure the proposal consistency.

Additionally we recommend including unit tests containing both correct and incorrect parameters in order to test the correct behavior in case of inconsistent inputs.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit d71149f68370940e45e198c7c77a99e9235e19af

GOV-05 | LOCKED BLOCKCHAIN NATIVE TOKENS

Category	Severity	Location	Status
Inconsistency	● Minor	packages/forge/src/governance/Governor.sol: 60	● Resolved

Description

The contract has a `receive()` function or payable functions, making it able to receive native tokens. However, it does not have a function to withdraw the funds, which can lead to permanently locked tokens within the contract.

```
60      receive() external payable virtual {
```

Recommendation

It is suggested to either remove the `receive()` function and the payable attribute, or add a withdraw function with proper access control mechanisms.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit 35d3fb25b58167dca1ac4168c23c1315b4cef4db by removing the `receive` function.

GSB-01 | POTENTIAL OUT-OF-GAS EXCEPTION

Category	Severity	Location	Status
Logical Issue	● Minor	packages/forge/src/governance/extensions/GovernorSorting.sol: 178	● Acknowledged

Description

When a loop allows an arbitrary number of iterations or accesses state variables in its body, the function may run out of gas and revert the transaction.

```
178         for (uint256 i = 0; i < _sortedProposalIds.length; i++) {
```

Function `Contest.setSortedAndTiedProposals` contains a loop and its loop condition depends on state variables: `_sortedProposalIds` .

Recommendation

We recommend imposing a loop bound to ensure the computation can be finalized or refactoring the logic to ensure that it will not meet an out of gas exception regardless of the size of `sortedProposalIds` .

Alleviation

[`CertiK`]: The team acknowledged the finding and decided to fix the issue in a future release.

GSB-04 | UNSAFE INTEGER CAST

Category	Severity	Location	Status
Incorrect Calculation	● Minor	packages/forge/src/governance/extensions/GovernorSorting.sol: 148, 184	● Resolved

Description

Type casting refers to changing an variable of one data type into another. The code contains an unsafe cast between integer types, which may result in unexpected truncation or sign flipping of the value.

```
148                 int256(proposalVoteCountsArray[i].forVotes) - int256(
proposalVoteCountsArray[i].againstVotes);
```

Casted expression `proposalVoteCountsArray[i].forVotes` has estimated range [0, 115792089237316195423570985008687907853269984665640564039457584007913129639935] but target type `int256` has range [-57896044618658097711785492504343953926634992332820282019728792003956564819968, 57896044618658097711785492504343953926634992332820282019728792003956564819967].

```
184             int256 currentTotalVotes = int256(currentForVotes) - int256(
currentAgainstVotes);
```

Casted expression `currentForVotes` has estimated range [0, 115792089237316195423570985008687907853269984665640564039457584007913129639935] but target type `int256` has range [-57896044618658097711785492504343953926634992332820282019728792003956564819968, 57896044618658097711785492504343953926634992332820282019728792003956564819967].

Recommendation

It is recommended to check the bounds of integer values before casting. Alternatively, consider using the `SafeCast` library from OpenZeppelin to perform safe type casting and prevent undesired behavior.

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/cf86fd9962701396457e50ab0d6cc78aa29a5ebc/contracts/utils/math/SafeCast.sol

Alleviation

[`CertiK`]: The team acknowledged the finding and fixed the issue in commit 5322dcb8fc9ecfaf36731c856139e234720592f1 by adopting the `SafeCast` library.

RMB-02 | POTENTIAL REENTRANCY ATTACK (OUT-OF-ORDER EVENTS)

Category	Severity	Location	Status
Concurrency	● Minor	packages/forge/src/modules/RewardsModule.sol: 233, 234, 288, 289, 302, 303	● Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

This finding is considered minor because the reentrancy only causes out-of-order events.

External call(s)

```
233         Address.sendValue(addressToPayOut, payment);
```

- This function call executes the following external call(s).
- In `Address.sendValue` ,
    - `(success,None) = recipient.call{value: amount}("")`

Events emitted after the call(s)

```
234         emit PaymentReleased(addressToPayOut, payment);
```

External call(s)

```
288         SafeERC20.safeTransfer(token, addressToPayOut, payment);
```

- This function call executes the following external call(s).
- In `SafeERC20._callOptionalReturn` ,
    - `returndata = address(token).functionCall(data,"SafeERC20: low-level call failed")`
- In `Address.functionCallWithValue` ,

  - `(success,returndata) = target.call{value: value}(data)`

Events emitted after the call(s)

```
        emit ERC20PaymentReleased(token, addressToPayOut, payment);
```

External call(s)

```
        SafeERC20.safeTransfer(token, payable(creator()), token.balanceOf(
address(this)));
```

- This function call executes the following external call(s).
- In `SafeERC20._callOptionalReturn` ,
  - `returndata = address(token).functionCall(data,"SafeERC20: low-level call failed")`
- In `Address.functionCallWithValue` ,
  - `(success,returndata) = target.call{value: value}(data)`

Events emitted after the call(s)

```
        emit ERC20RewardWithdrawn(token, creator(), token.balanceOf(address(
this)));
```

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit 3055f9a0b64928dcfaa2e55d06cabb1bfa101105 by adopting the check-effect-interact pattern.

RMB-03 | POTENTIAL DIVIDE BY ZERO

Category	Severity	Location	Status
Logical Issue	● Minor	packages/forge/src/modules/RewardsModule.sol: 315	● Resolved

Description

Performing division by zero would raise an error and revert the transaction.

```
315         return (totalReceived * _shares[ranking]) / _totalShares -
alreadyReleased;
```

The expression `(totalReceived * _shares[ranking]) / _totalShares` may divide by zero. Its divisor has has estimated interval [0, 231584178474632390847141970017375815706539969331281128078915168015826259279870].

Recommendation

It is recommended to either reformulate the divisor expression, or to use conditionals or require statements to rule out the possibility of a divide-by-zero. In particular, since the payees and shares lists can not be changed after the contract deployment, we recommend including a check on `_totalShares != 0` in the contract constructor.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit 73141af0616839ca0286c48fa7303dbf85398576 by adding the suggested `_totalShares != 0` check.

GMV-01 | MISSING PARAMETER COMMENT

Category	Severity	Location	Status
Inconsistency	● Informational	packages/forge/src/governance/GovernorMerkleVotes.sol: 33~37	● Resolved

Description

The `GovernorMerkleVotes.checkProof` methods has 4 paramters out of which only 3 are described in the developer comments.

Recommendation

We recommend including in the parameters description the `voting` parameter, too, so that code and documentation are aligned.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit 74401c2483fb731f33d4f5459192cec577332e78 by adding the requested comment.

GOV-06 | UNUSED CONSTANT

Category	Severity	Location	Status
Logical Issue	● Informational	packages/forge/src/governance/Governor.sol: 26	● Resolved

Description

The `BALLOT_TYPEHASH` constant in the `Governor` contract is used neither in the contract logic, nor in the contract extensions present in the audit scope.

Recommendation

We recommend reviewing the contract logic and evaluate the usefulness of the mentioned constant to double check that it does not refer to some missing or outdated logic. Then, in case it is not supposed to be used, we suggest removing it.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit 6f71b895736b6b77b0f85962f20d63f7965e2a75 by removing the unused constant.

GOV-07 | CREATOR CAN CANCEL AN ACTIVE CONTEST

Category	Severity	Location	Status
Logical Issue	● Informational	packages/forge/src/governance/Governor.sol: 334~348	● Resolved

Description

The logic deducted from the `Governor` code allows the `creator` account to cancel a vote while it is in the active state receiving participants votes.

This allows the creator to cancel a contest when, near to the deadline timestamp, the result is almost known.

Recommendation

The audit team wants to highlight the described logic and ask if this is the expected contract behavior.

Alleviation

[`CertiK`]: The team clarified that the intended behavior is grant such privilege to the `creator` account. Since this is the expected behavior, no action is required on the codebase.

GOV-08 | EMPTY PARAMETER IN `_castVote`

Category	Severity	Location	Status
Inconsistency	● Informational	packages/forge/src/governance/Governor.sol: 377, 394	● Resolved

Description

The `reason` parameter in the `_castVote` function is always hardcoded to be fulfilled with an empty string in the `Governor` contract.

Recommendation

We recommend reviewing the function parameter purpose and remove it if it is not intended to be used to store and log data.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit 079aadd074fe70670c5576441c3fd11a903d14cb by removing the `reason` parameter.

GSB-05 | COMPARISON TO BOOLEAN CONSTANT

Category	Severity	Location	Status
Coding Style	● Informational	packages/forge/src/governance/extensions/GovernorSorting.sol: 168~171	● Resolved

Description

Boolean constants can be used directly and do not need to be compared to true or false.

```
            require(
                setSortedAndTiedProposalsHasBeenRun == false,
"GovernorSorting: setSortedAndTiedProposals() has already been run and its
respective values set"

            );
```

Recommendation

We recommend removing the equality to the boolean constant.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit
87afbb6b98d533b8acf4cbe5961d5f8e42f0d3f9.

IGB-01 | HARDCODED ENUM COUNT

Category	Severity	Location	Status
Coding Style	● Informational	packages/forge/src/governance/IGovernor.sol: 19	● Resolved

Description

The `METADATAS_COUNT` constant contains the hardcoded count of the `Metadatas` enum possible cases. This implies a manual developer update each if `Metadatas` gets new enums added, which is a an error prone approach.

Recommendation

We recommend replacing the hardcoded value with a compile time computed value which is automatically updated when a new case is added to `Metadatas` .

An example is:

```
uint256 public constant METADATAS_COUNT = uint256(type(Metadatas).max) + 1;
```

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit c134e80c7996aa2241cfecd0600668e8f4541f68.

SRC-03 | MISSING ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	● Informational	packages/forge/src/governance/Governor.sol: 61, 250, 340; packages/forge/src/governance/extensions/GovernorModuleRegistry.sol: 28; packages/forge/src/modules/RewardsModule.sol: 293, 300	● Resolved

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit fe15ad3712a2323383fef97f09e150b38a1dc33d by providing the requested error messages.

SRC-04 | RANKING CALCULATION ON TIED PROPOSALS

Category	Severity	Location	Status
Logical Issue	● Informational	packages/forge/src/governance/extensions/GovernorSorting.sol: 163~229; packages/forge/src/modules/RewardsModule.sol: 219	● Resolved

Description

The `setSortedAndTiedProposals` method sorts proposals according to their collected votes and creates a ranking of them.

When computing the ranking, tied proposals are accounted for the same position and the following one has the subsequent position. As an example, if the first two proposals collected the same vote count, the subsequent proposal is counted as second in the ranking. However, since two proposals are before the subsequent one, it could also be accounted as third in the ranking and not as the second one.

The concern also arises from the revert message in the `RewardModule` contract, L219, where it explicitly says that ties are taken into account, while in the current logic they are not.

Recommendation

The audit team wants to highlight such logic to double check that the implemented code reflects the desired business requirement.

Alleviation

[`CertiK`]: The team clarified that the current ranking calculation implementation reflects the business logic.

OPTIMIZATIONS | JOKERACE

ID	Title	Category	Severity	Status
GMR-01	Unnecessary Storage Read	Gas Optimization	Optimization	● Resolved
GOE-01	User-Defined Getters	Gas Optimization	Optimization	● Acknowledged
GOV-01	Unnecessary Boolean As Return Value	Gas Optimization	Optimization	● Acknowledged
GOV-02	Inefficient Memory Parameter	Inconsistency	Optimization	● Resolved
GSB-02	Unnecessary Storage Read Access In For Loop	Coding Issue	Optimization	● Resolved
GSB-03	Useless Subtraction By 0	Gas Optimization	Optimization	● Resolved
GSB-06	Unnecessary Condition Check	Gas Optimization	Optimization	● Resolved
RMB-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	● Resolved
SRC-02	Costly Operation Inside Loop	Coding Issue	Optimization	● Acknowledged

GMR-01 | UNNECESSARY STORAGE READ

Category	Severity	Location	Status
Gas Optimization	● Optimization	packages/forge/src/governance/extensions/GovernorModuleRegistry.sol: 31	● Resolved

Description

The `setOfficialRewardsModule` function sets a new value for `_officialRewardsModule` and emits the related event. When emitting the event, `_officialRewardsModule` is read again from the storage. However, its value is the same as `officialRewardsModule_` which, instead, is in memory so reading it is cheaper from the gas point of view.

Recommendation

We recommend reading the pointed value from memory, that is cheaper for gas optimization.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit e339d77469de3a9433b80641ff83b367f9d7a3b1.

GOE-01 | USER-DEFINED GETTERS

Category	Severity	Location	Status
Gas Optimization	● Optimization	packages/forge/src/governance/Governor.sol: 187~192, 194~199, 201~206; packages/forge/src/governance/extensions/GovernorCountingSimple.sol: 77~84	● Acknowledged

Description

The linked functions are equivalent to the compiler-generated getter functions for the respective variables.

Recommendation

We advise removing the pointed getters and relying on the compiler-generated getter functions for accessing the linked variables as they are less prone to error and much more maintainable than manually written ones.

Alleviation

[`CertiK`]: The team acknowledged the finding and decided to remain unchanged.

GOV-01 | UNNECESSARY BOOLEAN AS RETURN VALUE

Category	Severity	Location	Status
Gas Optimization	● Optimization	packages/forge/src/governance/Governor.sol: 235, 350 ~357	● Acknowledged

Description

The `validateProposalData` validates the correctness of an input proposal and reverts in case an inconsistency is found. As a result, the output `dataValidated` parameter only returns `true` in a successful execution, which is unnecessary since there is not any information content in such return value.

Similarly, the function `verifyVoter` either returns true or reverts.

Recommendation

We recommend to either remove the unnecessary return values or to use them and then check the result in the caller, reverting if needed.

Alleviation

[`CertiK`]: The team acknowledged the finding and decided to remain unchanged.

GOV-02 | INEFFICIENT MEMORY PARAMETER

Category	Severity	Location	Status
Inconsistency	● Optimization	packages/forge/src/governance/Governor.sol: 262, 276, 313	● Resolved

Description

One or more parameters with `memory` data location are never modified in their functions and those functions are never called internally within the contract. Thus, their data location can be changed to `calldata` to avoid the gas consumption copying from calldata to memory.

```
174      function propose(ProposalCore memory proposal, bytes32[] calldata proof)
```

`propose` has memory location parameters: `proposal` .

```
185      function proposeWithoutProof(ProposalCore memory proposal) public virtual
returns (uint256 proposalId);
```

`proposeWithoutProof` has memory location parameters: `proposal` .

```
313      function deleteProposals(uint256[] memory proposalIds) public virtual {
```

`deleteProposals` has memory location parameters: `proposalIds` .

Recommendation

We recommend changing the parameter's data location to `calldata` to save gas.

- For Solidity versions prior to 0.6.9, since public functions are not allowed to have calldata parameters, the function visibility also needs to be changed to `external` .
- For Solidity versions prior to 0.5.0, since parameter data location is implicit, changing the function visibility to `external` will change the parameter's data location to calldata as well.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit d7a9eb8740afe9692d81c1d2afe2d52f20264dff by using the `calldata` data location.

GSB-02 | UNNECESSARY STORAGE READ ACCESS IN FOR LOOP

Category	Severity	Location	Status
Coding Issue	● Optimization	packages/forge/src/governance/extensions/GovernorSorting.sol: 178	● Resolved

Description

The for loop contains repeated storage read access in the condition check. Given that the ending condition does not change in the for loop, the repeated storage read is unnecessary, and its associated high gas cost can be eliminated.

```
178         for (uint256 i = 0; i < _sortedProposalIds.length; i++) {
```

Loop condition `i < _sortedProposalIds.length` accesses the `length` field of a storage array.

Recommendation

Storage access costs substantially more gas than memory and stack access. We recommend caching the variable used in the condition check of the for loop to avoid unnecessary storage access.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit e5bb7bd145a4f54a51b6691849ffc27e852ed424.

GSB-03 | USELESS SUBTRACTION BY 0

Category	Severity	Location	Status
Gas Optimization	● Optimization	packages/forge/src/governance/extensions/GovernorSorting.sol: 193	● Resolved

Description

The code at the pointed line subtract `i` to `lastSortedItemIndex`. However the whole code block is executed only if `i == 0`, which makes useless the highlighted subtraction.

Recommendation

We recommend removing the useless operation in order to save gas and improve the code clarity,

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit 21d3e618bb0aeafb124fbaf18d44b5adbf5a5467.

GSB-06 | UNNECESSARY CONDITION CHECK

Category	Severity	Location	Status
Gas Optimization	● Optimization	packages/forge/src/governance/extensions/GovernorSorting.sol: 213	● Resolved

Description

The pointed condition checks `currentTotalVotes != lastTotalVotes` right after an if construct checking the opposite condition `currentTotalVotes == lastTotalVotes` .

Recommendation

We recommend saving the second if construct by extending the previous one with an `else` block.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit 2fc7b793cf7eeeeff3c03061de2c92e603be2a50.

RMB-01 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	● Optimization	packages/forge/src/modules/RewardsModule.sol: 48, 49	● Resolved

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

[`CertiK`]: The team acknowledged the finding and solved the issue in commit 4475384b0ceab6cb75988c5567b7355b03301b22 by declaring the variables as immutable.

SRC-02 | COSTLY OPERATION INSIDE LOOP

Category	Severity	Location	Status
Coding Issue	● Optimization	packages/forge/src/governance/extensions/GovernorSorting.sol: 194, 208, 222; packages/forge/src/modules/RewardsModule.sol: 330	● Acknowledged

Description

Reading, initializing, and modifying storage variables cost more gas than operating local variables, and this gas cost can significantly increase when these operations are performed inside a loop.

Reference: https://docs.soliditylang.org/en/latest/introduction-to-smart-contracts.html#storage-memory-and-the-stack

```
208    _highestTiedRanking = rankingBeingChecked;
```

```
330    _totalShares = _totalShares + shares_;
```

called at

```
68    _addPayee(payees[i], shares_[i]);
```

Recommendation

It is suggested to use a local variable to hold the loop computation result, reducing gas consumption and improving the contract's efficiency. Then, result can be set when the loop ended.

Alleviation

[`CertiK`]: The team acknowledged the finding and decided to remain unchanged.

APPENDIX | JOKERACE

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Concurrency	Concurrency findings are about issues that cause unexpected or unsafe interleaving of code executions.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED “AS IS” AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.