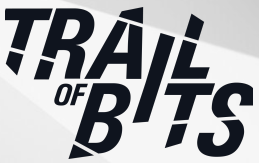

Ink Foundation Governance System

Security Assessment (Summary Report)

June 12, 2025

Prepared for:
Ink Foundation

Prepared by: **Priyanka Bose, Sam Alws, Jaime Iglesias, and Nicolas Donboly**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Sam Greenup, Project Manager
sam.greenup@trailofbits.com

The following engineering director was associated with this project:

Benjamin Samuels, Engineering Director, Blockchain
benjamin.samuels@trailofbits.com

The following consultants were associated with this project:

Priyanka Bose, Consultant
priyanka.bose@trailofbits.com

Sam Alws, Consultant
sam.alws@trailofbits.com

Jaime Iglesias, Consultant
jaime.iglesias@trailofbits.com

Nicolas Donboly, Consultant
nicolas.donboly@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
May 8, 2025	Pre-project kickoff call
May 16, 2025	Delivery of report draft
May 16, 2025	Report readout meeting
June 12, 2025	Delivery of final summary report

Project Targets

The engagement involved reviewing and testing the following target.

ungovernable

Repository	https://github.com/inkfoundation/inktoken
Version	`585d3b5df2eb4648857ca0f4544a7d1af304cf4c`
Type	Solidity
Platform	Ethereum

Executive Summary

Engagement Overview

Ink Foundation engaged Trail of Bits to review the security of its governance system, which provides administrative control before transitioning to decentralized governance. The system consists of an ERC-20 token with specialized governance capabilities and a configurable governor contract built on top of OpenZeppelin's governance framework.

One consultant conducted the review from May 12 to May 16, 2025, with three consultants shadowing, for a total of one engineer-week of effort. We focused on manually reviewing the smart contracts for potential vulnerabilities in the governance mechanisms, the permission controls, and the security of the transition from centralized to decentralized governance. With full access to source code and deployment scripts, we performed static analysis of the codebase through manual review processes.

Observations and Impact

The scope of this review focused on the governance contracts, which implement a two-phase governance model that transitions from centralized control to decentralized on-chain governance. Our analysis evaluated the system's core architectural elements—including role-based access controls, transfer restrictions, voting mechanisms, and governance parameter configuration; we also conducted a comprehensive assessment of the system's permissions management, the ownership transition process, and the security of the administrative functions. We also examined the inherited OpenZeppelin libraries to verify that they are properly implemented and to identify any known vulnerabilities, missing features, or security gaps in the integration.

Overall, the governance implementation demonstrates a thoughtful approach to enabling controlled governance transitions in token ecosystems. We identified four informational-severity issues: toggle-based functions that are problematic for governance voting (TOB-INK-1), insufficient documentation regarding the absence of critical features like a timelock (TOB-INK-2), insecure default governance parameter configurations with low thresholds (TOB-INK-3), and unclear initial token distribution flow (TOB-INK-4). Additional concerns include missing event emissions for administrative actions, inconsistent use of bits to indicate system roles, and minor code quality issues, all listed in appendix C. While these issues do not compromise the system's fundamental security, they affect its usability and stability. Addressing these concerns would enhance the robustness and reliability of the governance system.

Recommendations

Based on the findings identified in the security review, Trail of Bits recommends that Ink Foundation take the following actions:

- **Fix the disclosed findings.** Address the issues identified in this report to improve the consistency and reliability of the governance system implementation.
- **Enhance governance parameter validation.** Implement minimum thresholds for critical governance parameters, particularly the quorum percentage, proposal threshold, and voting delay parameters to prevent governance manipulation.
- **Document security considerations.** Explicitly document the absence of critical features like timelock controllers and explain the security implications and mitigations for users and integrators.
- **Incorporate deployment script testing.** Incorporate comprehensive tests for the deployment scripts to ensure secure contract initialization and governance transition. The tests provided in appendix D can offer an effective foundation for building this testing.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The contracts do not contain any arithmetic operations, mostly relying on arithmetic performed in OpenZeppelin's libraries.	**Not Applicable**
Auditing	While the inherited OpenZeppelin libraries provide sufficient event emissions for standard operations, the custom implementation lacks comprehensive events for critical administrative actions unique to this system, particularly blacklisting and whitelist management functions that would benefit from additional transparency.	**Moderate**
Authentication / Access Controls	The system implements a robust role-based access control mechanism using the `OwnableRoles` contract from Solady. There is clear separation between owner and admin role capabilities, with well-defined permissions for transfer control and blacklisting. The governance transition process has proper authorization checks.	**Satisfactory**
Complexity Management	The contracts inherit most functionality from OpenZeppelin libraries with minimal custom logic. The codebase is concise and follows a clear separation of concerns between token and governance functionalities. The inheritance structure is logical and well organized.	**Strong**
Cryptography and Key Management	The contracts delegate cryptography responsibilities to inherited libraries rather than implementing these functions directly. For deployment operations, the scripts use a `PRIVATE_KEY` environment variable stored in a `.env` file accessed via the `dotenv` command. While this approach satisfies basic deployment needs, a more	**Moderate**

	robust security practice would involve storing this sensitive key in a hardware wallet or dedicated secret management solution. It should be noted that this potential security concern is mitigated by the limited scope of the key's usage—it is employed only during the initial deployment phase and carries no persistent permissions after the deployment process completes.	
Decentralization	Once deployment, token distribution, and governance transition are complete, the system will transition to being fully governance-owned—no entity retains privileged access, as all administrative functions transfer to the on-chain governance mechanism. Note that the "degree of decentralization" will mostly depend on governance token distribution.	**Satisfactory**
Documentation	The codebase uses NatSpec format inconsistently. While function parameters and error conditions are documented, some functions lack comprehensive descriptions.	**Moderate**
Low-Level Manipulation	A small amount of assembly is used in the `UngovernableERC20` contract to perform reverts; the rest of the assembly comes from standard libraries. We did not find any issues with this assembly code.	**Satisfactory**
Testing and Verification	Based on our review of the provided materials, the core `UngovernableERC20` and `UngovernableGovernor` contracts appear to have comprehensive test coverage. However, we found no evidence of testing for the deployment scripts. Testing of deployment scripts is crucial for ensuring secure contract initialization and governance transition. Test files similar to those referenced in appendix D would significantly improve overall verification coverage and strengthen the system's reliability.	**Satisfactory**
Transaction Ordering	The governance system provides adequate protection against most transaction ordering attacks through voting delay and period mechanisms. However, the absence of a timelock controller means successful proposals can be executed immediately after they are accepted, which could create governance manipulation risks if multiple related proposals are passed simultaneously.	**Moderate**

Summary of Findings

The table below summarizes the findings of the review, including details on type and severity.

ID	Title	Type	Severity
1	toggleBlacklist and toggleWhitelist functions are difficult to vote on	Configuration	**Informational**
2	Lack of explicit documentation of key features the governor does not implement	Documentation	**Informational**
3	Insufficient governance parameter validation could lead to governance instability	Data Validation	**Informational**
4	Unclear token distribution flow	Undefined Behavior	**Informational**

Detailed Findings

1. toggleBlacklist and toggleWhitelist functions are difficult to vote on	
Severity: **Informational**	Difficulty: **Low**
Type: Configuration	Finding ID: TOB-INK-1
Target: `src/UngovernableERC20.sol`	

Description
The `toggleBlacklist` and `toggleWhitelist` functions, which are used to add or remove users from the blacklist/whitelist, are difficult to vote on and should be replaced with functions that take a Boolean indicating whether to add or remove a user from the list.

For example, if two proposals to perform a `toggleBlacklist` call on the same user were being voted on at the same time, it would be difficult for a voter to decide which one to vote for. If both proposals are approved, the two calls to `toggleBlacklist` on the same user would cancel each other out.

This can expand out into more complicated scenarios that are even more difficult to vote on, such as scenarios where there is one proposal to blacklist five users at once and another proposal to blacklist only one of those users.

If these functions were replaced with `setBlacklist(user,bool)` and `setWhitelist(user,bool)` functions, the double-voting issue could not occur: even if two separate proposals to call `setBlacklist(user,true)` are approved, the user will still be blacklisted.

Recommendations
Short term, replace the `toggleBlacklist` and `toggleWhitelist` functions with functions that take a Boolean indicating whether the user should be added or removed from the list.

Long term, whenever implementing features that can be used through a governance mechanism, make them as explicit as possible to prevent situations in which multiple proposals being executed can lead to unexpected behavior.

2. Lack of explicit documentation of key features the governor does not implement

Severity: **Informational**	Difficulty: **Low**
Type: Documentation	Finding ID: TOB-INK-2
Target: `src/UngovernableGovernor.sol`	

Description

The system lacks documentation on some of the features the governor does not implement, such as the following:

- There is no timelock, which means there is no delay between proposal approval and execution.
- There are no guardians or similar roles, so approved proposals are final. This might seem obvious, but highlighting that there is no "mitigation mechanism" for malicious or "buggy" proposals is important, as it highlights how critical proper proposal management and active governance are.

Documenting the absence of these features will help inform users of their responsibilities and the safeguards the system has.

Finally, referencing OpenZeppelin's own governance documentation would be useful for users to quickly understand the core features of the governor.

Recommendations

Short term, add documentation on the absence of features in the governor system, such as the lack of a timelock mechanism and guardians.

Long term, whenever implementing a governance mechanism, consider what features can be considered "standard," are widely adopted by "popular" governance systems, or might be expected by users, and make sure to document whether those features are implemented to help inform users.

3. Insufficient governance parameter validation could lead to governance instability

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-INK-3
Target: `deploy.config.json`	

Description

The governance system uses default configuration parameters that present risks due to its minimalist architecture lacking timelocks and guardian mechanisms, potentially leading to governance instability.

For example, the `_initialQuorumPercentage` parameter is at only 4%, the `_initialProposalThreshold` parameter is set to only 1 token, and the `_initialVotingDelay` parameter is set to only 1 day. In this minimal governance system design without safeguards such as timelocks and guardians, where proposals execute immediately upon passing, such low thresholds create practical governance challenges. The 4% quorum allows decisions to be made by a relatively small token minority; the 1-token proposal threshold could lead to numerous proposals requiring community attention; and the 1-day voting window may be insufficient for thorough deliberation on complex changes. These settings require consistently active governance participation to maintain system security, which may be difficult to sustain in a decentralized community.

Recommendations

Short term, increase the governance parameter values to more secure levels: raise the quorum percentage (e.g., greater than 4%) if a timelock is not set, set a higher proposal threshold (e.g., 0.5 to 1% of token supply), and extend the voting delay (e.g., 2 to 3 days) to allow adequate discussion time.

Long term, consider adding a timelock mechanism for sensitive governance actions to provide additional security against governance attacks.

4. Unclear token distribution flow	
Severity: **Informational**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-INK-4
Target: `src/UngovernableERC20.sol`	

Description
There are no scripts that can be used to mint tokens or whitelist users during deployment. This necessary part of deployment would need to be done ad hoc, increasing the chance of a mistake and decreasing the project's auditability.

In addition, if the standard deployment flow is followed, nobody will have permission to whitelist users during deployment, since the `toggleWhitelist` function is guarded by `onlyRoles(DEFAULT_ADMIN_ROLE)` and the admin role is not granted until right before ownership is renounced.

Exploit Scenario
Alice deploys a new `UngovernableERC20` and `UngovernableGovernor`. To perform the initial token distribution, she deploys a contract to which she mints the initial token supply, expecting users to be able to claim their tokens from it; however, with her current permissions, she is unable to whitelist the contract for the distribution to happen. In order to whitelist the contract, she needs to give herself admin permissions, whitelist the contract, and then remove her admin permissions.

Recommendations
Short term, change the `toggleWhitelist` function so that it is guarded by `onlyRolesOrOwner` instead of `onlyRoles`, so that the owner has permission to whitelist users during distribution.

Long term, ensure that all major actions that need to be taken during distribution have corresponding scripts and documentation explaining how they can be performed. Also, consider writing token distribution scripts.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.

Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category does not apply to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Code Quality Recommendations

This appendix contains recommendations for findings that do not have immediate or obvious security implications. However, these issues may facilitate exploit chains targeting other vulnerabilities, become easily exploitable in future releases, or decrease code readability.

- **Use a single bit to indicate the default admin role.** The `OwnableRoles` contract uses single-bit `uint256` values to indicate single roles, and multi-bit values to indicate multiple roles. For example, `0110` would indicate that a user has both the `0100` and `0010` roles. The `DEFAULT_ADMIN_ROLE` constant defined in the `UngovernableERC20` contract, which inherits from `OwnableRoles`, has many bits set even though it is used to indicate a single role. It should be replaced with a single-bit value such as `OwnableRoles._ROLE_0`, which equals `1`.

```
/// @dev The role for the DEFAULT_ADMIN_ROLE, controls enabling transfers and
adding/removing addresses from the blacklist
uint256 public constant DEFAULT_ADMIN_ROLE =
uint256(keccak256("DEFAULT_ADMIN_ROLE"));
```

Figure C.1: Definition of the `DEFAULT_ADMIN_ROLE` constant (`src/UngovernableERC20.sol:18–19`)

- **Add event emissions to functions that modify state:**
  - `src/UngovernableERC20.sol#L29–L32`
  - `src/UngovernableERC20.sol#L34–37`
  - `src/UngovernableERC20.sol#L70–L73`
  - `src/UngovernableERC20.sol#L75–L78`
- **Remove the following unused imports:**
  - `src/UngovernableERC20.sol#L4`
  - `src/UngovernableGovernor.sol#L9`
  - `src/UngovernableGovernor.sol#L12–L13`
- **Remove the following unused submodule:**
  - `.gitmodules#L10–L12`
- **Correct the following typos:**
  - In the following comment, `blackisting` should be `blacklisting`:

        - src/UngovernableERC20.sol#L9
    - In the following comment, `overidden` should be `overridden`:
        - src/UngovernableERC20.sol#L40
- **Correct the comment linked below.** This comment suggests there is a burn mechanism; however, the client informed us that this was left from a previous design. The comment should be fixed to only mention minting to avoid confusion.
    - src/UngovernableERC20.sol#L41

D. Test Files for Deployment Scripts

We noticed that neither of the deployment scripts was covered by the test suite, so we provided test files for each one (see figures D.1 and D.2), raising the test coverage. We will send both test files separately from this report.

```
function test_Deployment_Properties() public view {
    // Assertions for token and governor properties against expected configuration
values
    assertEq(token.owner(), deployerAddress, "Token owner should be the deployer of
the script");
    assertEq(address(governor.token()), address(token), "Governor's token should be
the deployed token instance");

    assertEq(token.name(), expectedTokenName, "Token name should match config");
    assertEq(token.symbol(), expectedTokenSymbol, "Token symbol should match
config");

    assertEq(governor.name(), expectedGovernorName, "Governor name should match
config");
    assertEq(governor.proposalThreshold(), expectedInitialProposalThreshold,
"Governor proposalThreshold should match config's _initialProposalThreshold");
    assertEq(governor.quorumNumerator(), expectedInitialQuorumPercentage, "Governor
quorumNumerator should match config's _initialQuorumPercentage");
    assertEq(governor.votingDelay(), uint256(expectedInitialVotingDelay), "Governor
votingDelay should match config's _initialVotingDelay");
    assertEq(governor.votingPeriod(), uint256(expectedInitialVotingPeriod),
"Governor votingPeriod should match config's _initialVotingPeriod");
}
```

Figure D.1: Snippet of the test file `DeployScriptTest.t.sol`

```
function test_TokenOwner_Is_AddressZero() public view {
    assertEq(token.owner(), address(0), "Token owner should be the address(0) after
renunciation.");
}

function test_Governor_Has_DefaultAdminRole_For_Token() public view {
    assertTrue(
        token.hasAnyRole(address(governor), adminRole),
        "Governor should have DEFAULT_ADMIN_ROLE on the Token contract."
    );
}
```

Figure D.2: Snippet of the test file `RenounceToGovernanceScriptTest.t.sol`

E. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On May 22, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the Ink Foundation team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, Ink Foundation has resolved all four issues described in this report. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	toggleBlacklist and toggleWhitelist functions are difficult to vote on	Informational	Resolved
2	Lack of explicit documentation of key features the governor does not implement	Informational	Resolved
3	Insufficient governance parameter validation could lead to governance instability	Informational	Resolved
4	Unclear token distribution flow	Informational	Resolved

Detailed Fix Review Results

TOB-INK-1: toggleBlacklist and toggleWhitelist functions are difficult to vote on
Resolved in PR #2. This PR replaces the `toggleBlacklist` and `toggleWhitelist` functions with `setBlacklist` and `setWhitelist` functions, as recommended in the detailed finding.

TOB-INK-2: Lack of explicit documentation of key features the governor does not implement
Resolved in PR #2. This PR adds a section to the project's README file stating that the governor does not have timelock or guardian features and explaining why this design was chosen.

TOB-INK-3: Insufficient governance parameter validation could lead to governance instability
Resolved in PR #2. This PR edits the `deploy.config.json` file, adjusting the quorum percentage to 5% and the minimum proposal tokens to 10 million tokens. It also updates the README file to reflect these changes.

TOB-INK-4: Unclear token distribution flow
Resolved in PR #2. This PR changes the `enableTransfer`, `setWhitelist`, and `setBlacklist` functions so that they are guarded by `onlyRolesOrOwner(DEFAULT_ADMIN_ROLE)` instead of `onlyRoles(DEFAULT_ADMIN_ROLE)`.

F. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review assessments, supporting client organizations in the technology, defense, blockchain, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, Uniswap, Solana, Ethereum Foundation, Linux Foundation, and Zoom.

To keep up to date with our latest news and announcements, please follow @trailofbits on X or LinkedIn, and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.