RESONANCE

hemi

Hemi
Hemi Token Smart Contract Audit Report
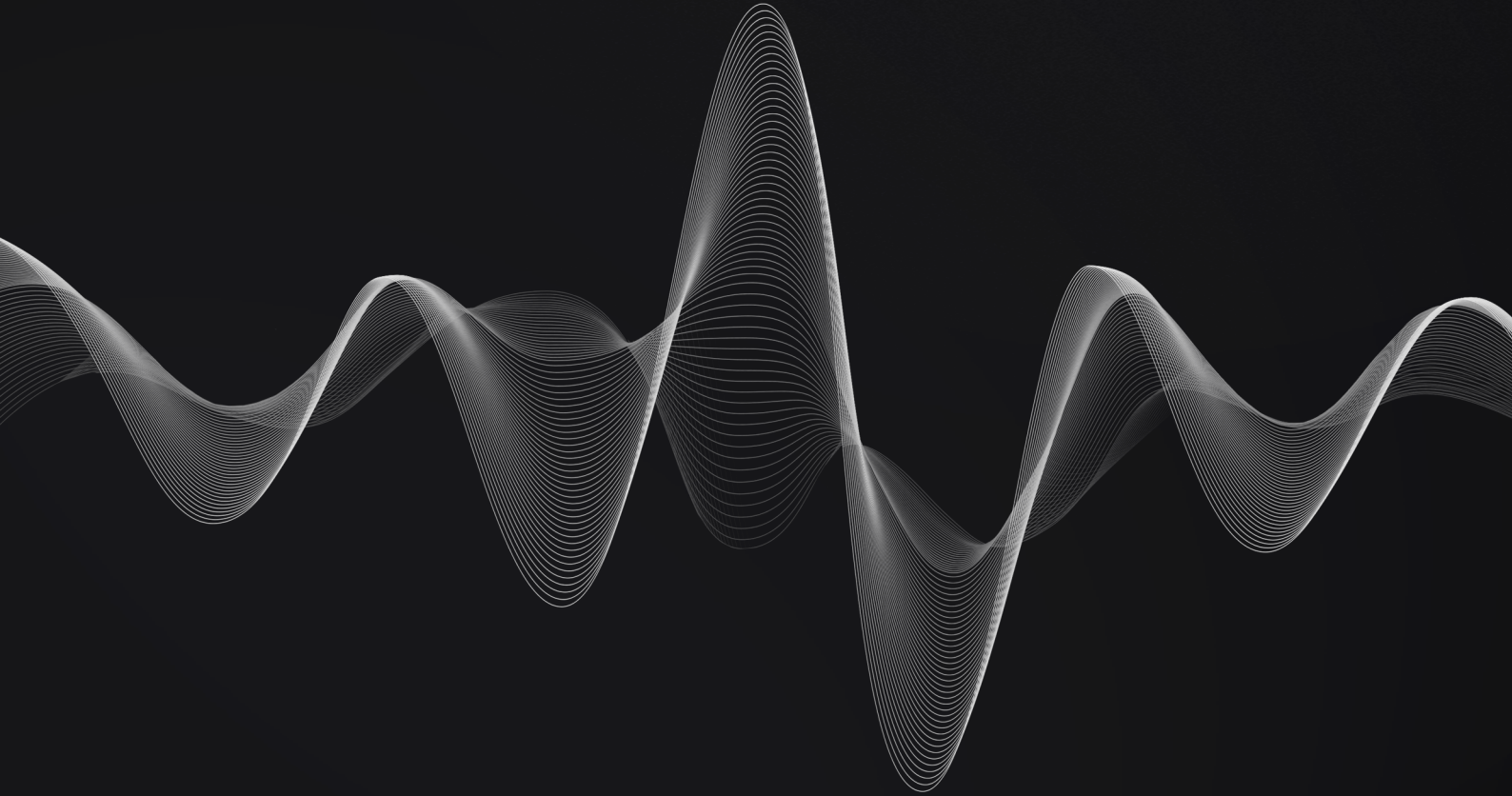
Document Control

FINAL(v2.0)

Audit_Report_HEMI-TOK_FINAL_20

Apr 15, 2025 ○	v0.1	Luis Arroyo: Initial draft
Apr 16, 2025 ○	v0.2	Luis Arroyo: Added findings
Apr 16, 2025 ●	v1.0	Charles Dray: Approved
Apr 23, 2025 ○	v1.1	Luis Arroyo: Reviewed findings
Apr 25, 2025 ●	v2.0	Charles Dray: Finalized

Points of Contact	Max	Hemi	max@hemi.xyz
	Charles Dray	Resonance	charles@resonance.security
Testing Team	Luis Arroyo	Resonance	luis.arroyo@resonance.security
	João Simões	Resonance	joao@resonance.security
	Michał Bazyli	Resonance	michal@resonance.security

Copyright and Disclaimer

Contents

Executive Summary

Hemi contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between 14th April, 2025 and 16th April, 2025. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 2 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 3 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide Hemi with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.

System Overview

The Hemi token is a ERC20 token with governance and gasless approval capabilities, initially minting 10B tokens. It implements monthly token emissions based on configurable annual inflation rate and automatically bridges all newly tokens to L2 through a tunnel system with configurable destination addresses.

Repository Coverage and Quality

Code	Tests	Documentation
8 / 10	9 / 10	5 / 10
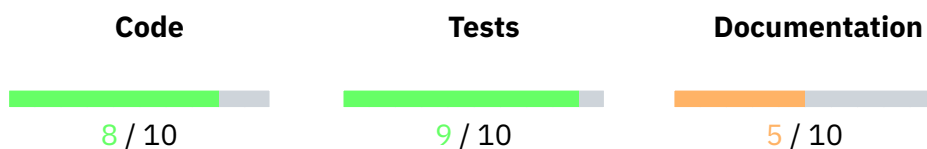
Resonance's testing team has assessed the Code, Tests, and Documentation coverage and quality of the system and achieved the following results:

- The code follows development best practices and makes use of known patterns, standard libraries, and language guides. It is easily readable and uses the latest stable version of relevant components. Overall, **code quality is, good**.

- Unit and integration tests are included. The tests cover both technical and functional requirements. Code coverage is 90%. Overall, **tests coverage and quality is excellent**.

- The documentation only includes the specification of the system. Overall, **documentation coverage and quality is average**.

© 2025 Resonance Security, Inc

Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: `hemilabs/hemi-token`

- Hash: 8e3988b6ad1372ce0306161bd6418bb8615dc645

The following items are excluded:

- External and standard libraries

- Files pertaining to the deployment process

- Financial related attacks

Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

Source Code Review - Solidity EVM

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities

2. Assert functionalities work as intended and specified

3. Deploy system in test environment and execute deployment processes and tests

4. Perform automated code review with public and proprietary tools

5. Perform manual code review with several experienced engineers

6. Attempt to discover and exploit security-related findings

7. Examine code quality and adherence to development and security best practices

8. Specify concise recommendations and action items

9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Solidity EVM audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Reentrancy attacks

- Frontrunning attacks

- Unsafe external calls

- Unsafe third party integrations

- Denial of service

- Access control issues

© 2025 Resonance Security, Inc

- Inaccurate business logic implementations

- Incorrect gas usage

- Arithmetic issues

- Unsafe callbacks

- Timestamp dependence

- Mishandled panics, errors and exceptions

Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss

2. Medium - Temporary or partial damage or loss

3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions

2. Likely - Requires technical knowledge or no special conditions

3. Very Likely - Requires trivial knowledge or effort or no conditions

Likelihood

	Very Likely	Likely	Unlikely
Strong	Critical	High	Medium
Medium	High	Medium	Low
Weak	Medium	Low	Info

Impact

© 2025 Resonance Security, Inc

Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.

- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.

- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

"Quick Win" Requires little work for a high impact on risk reduction.

"Standard Fix" Requires an average amount of work to fully reduce the risk.

"Heavy Project" Requires extensive work for a low impact on risk reduction.

RES-01	annualInflationRate Could Block Protocol If Set To 0		Resolved
RES-02	annualInflationRate Does Not Have Upper or Lower Limit		Resolved
RES-03	Lack of Token Distribution Leads to Centralization Problems		Acknowledged
RES-04	Centralization Risk For Trusted Owners		Acknowledged
RES-05	Update Import Usages To Add Modularity		Resolved
RES-06	Floating Pragma		Resolved

annualInflationRate Could Block Protocol If Set To 0

Code Section

- src/Hemi.sol#L134

Description

The global variable `annualInflationRate` is used to adjust the amount when new tokens are minted after the last issuance.

The owner can only adjust the value downwards, as it is not allowed to use a higher current value.

In case the owner mistakenly sets 0 as the value of this variable, it would not be possible to mint new tokens, since `_emissionAmount` in `calculateEmission()` would always be 0.

Recommendation

It is recommended to add a measure to avoid blocking minting of new tokens if `annualInflationRate` becomes 0.

Status

The issue has been fixed in commit f267afb850451fa1ebeede4632af296b822bfa0d.

annualInflationRate Does Not Have Upper or Lower Limit

Medium   **RES-HEMI-TOK02**   Data Validation   **Resolved**

Code Section
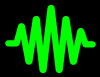
- `src/Hemi.sol#L55`

Description

The `annualInflationRate` does not have any upper or lower limit in the `constructor()`. This may block the protocol or cause a bad experience for users if an incorrect value is indicated.

Recommendation

It is recommended to add lower and upper limits to variables that affect to the amount minted tokens to avoid inappropiate behavior of the protocol.

Status

The issue has been fixed in commit f267afb850451fa1ebeede4632af296b822bfa0d.

Lack of Token Distribution Leads to Centralization Problems

Low   **RES-HEMI-TOK03**   Business Logic   **Acknowledged**

Code Section
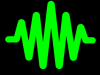
- src/Hemi.sol#L57

Description

If an address holds most of the total supply of a token, potential consequences arise. Centralization problems would appear, since a small group controls a large portion of the tokens. Market manipulation problems could emerge too, because volatility could be artificially increased. Finally, there would be a decrease in trust and confidence in the project if the tokens are not distributed more equitably.

Recommendation

If it is not possible to mint new tokens by the users and only one address will be in charge of distributing them, it is recommended to add to documentation the distribution system that will be followed by the protocol.

Status

The issue has been marked as acknowledged with the comment "There will be documentation on it".

© 2025 Resonance Security, Inc

Centralization Risk For Trusted Owners

RES-HEMI-TOK04   Access Control   **Acknowledged**

Code Section

- src/Hemi.sol

Description

The protocol contains several functions access controlled by owners with privileged rights in charge of performing admin tasks like setting the proxy or setting fees. These owners need to be trusted to not perform malicious updates on the protocol.

Recommendation

It is recommended to implement solutions like a multi-signature wallet to distribute admin control among multiple trusted parties. This ensures that critical actions can only be executed if a pre-defined quorum of trusted parties approves the action, reducing the risk of unilateral decisions or key compromise.

Status

The issue has been marked as acknowledged with the comment "There will be multisig owner".

RES-HEMI-TOK05   Code Quality   **Resolved**

Code Section

- src/Hemi.sol

Description

Two different syntaxes for importing contracts from external libraries or dependencies are used in Solidity:

1. `import "@openzeppelin/contracts/token/ERC20/ERC20.sol";`

2. `import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol"`

Second syntax allows developers to selectively import specific contracts from a file and provides finer control over which contracts are being imported. This could be useful for reducing namespace clutter or importing only necessary code for the contract.

Recommendation

It is recommended to update imports with curly braces usage if possible.

Status

The issue has been fixed in commit f267afb850451fa1ebeede4632af296b822bfa0d.

Floating Pragma

RES-HEMI-TOK06   Code Quality   **Resolved**

Code Section

- src/Hemi.sol

Description

The project uses floating pragmas `^0.8.28`.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Recommendation

It is recommended to use a strict and locked pragma version for solidity code. Preferably, the version should be neither too new or too old.

Status

The issue has been fixed in commit f267afb850451fa1ebeede4632af296b822bfa0d.

Proof of Concepts

No Proof-of-Concept was deemed relevant to describe findings in this engagement.