Quantstamp

Nillion - Smart Contracts

Executive Summary

This security review was prepared by Quantstamp, the leader in blockchain security.

Type	ERC-20 Token
Timeline	2025-10-20 through 2025-10-21
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• NillionNetwork/nil-erc20-token #af14741
Auditors	• Tim Sigl Auditing Engineer

Documentation quality	Undetermined
Test quality	Undetermined
Total Findings	7 Unresolved: 7
High severity findings	0
Medium severity findings	0
Low severity findings	5 Unresolved: 5
Undetermined severity findings	0
Informational findings	2 Unresolved: 2

Summary of Findings

Nillion implements an upgradeable ERC-20 token with minting and burning capabilities. The architecture is simple and builds on established OpenZeppelin contracts. We found no high or medium severity issues, only a few low severity findings. We would recommend establishing a test suite covering initialization, role permissions, mint/burn flows, and upgrade paths so production changes remain verifiable.

ID	DESCRIPTION	SEVERITY	STATUS
NIL-1	Unencrypted Private Key Storage in Deployment Scripts	Low	Unresolved
NIL-2	Centralized Power	Low	Unresolved
NIL-3	Missing Storage Gap Introduces Storage Collision Possibility	Low	Unresolved
NIL-4	Unsafe Access Control Not Following the Two-Step Pattern	Low	Unresolved
NIL-5	Unusual Use of the `initializer` Modifier Could Leave Implementation Partially Unlocked	Low	Unresolved
NIL-6	Outdated Solidity Version	Informational	Unresolved
NIL-7	No Test Suite	Informational	Unresolved

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> **Disclaimer**
>
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

Repo: `https://github.com/NillionNetwork/nil-erc20-token`

Included Paths: `src/` , `script/`

Operational Considerations

- Mint and burn only in line with the documented issuance policy to prevent unauthorized supply changes.
- Fully test and review new implementations before calling `upgradeTo()` to avoid introducing faulty or malicious logic.

Key Actors And Their Capabilities

Default Admin (DEFAULT_ADMIN_ROLE)

Responsibilities

- Manages role assignments through `grantRole()` / `revokeRole()` .
- Can execute any function of the roles below.

Trust Assumptions

- Admin key must remain uncompromised; any loss lets an attacker reassign roles, mint supply, or push malicious upgrades.

Minter (MINTER_ROLE)

Responsibilities

- Calls `mint()` to mint new tokens.

Trust Assumptions

- Holder must follow issuance policy; compromise or misuse directly inflates supply.

Burner (BURNER_ROLE)

Responsibilities

- Executes `burn()` to burn tokens from specified addresses.

Trust Assumptions

- Holder must follow issuance policy; compromise or misuse directly deflates supply.

Upgrader (UPGRADER_ROLE)

Responsibilities

- Approves `_authorizeUpgrade()` so the proxy can upgrade to new implementations.

Trust Assumptions

- Holder must validate deployments; a compromised upgrader can upgrade existing contracts to malicious contracts.

Findings

NIL-1 Unencrypted Private Key Storage in Deployment Scripts

• Low Unresolved

File(s) affected: `DeployNillionImpl.s.sol`, `UpgradeNillion.s.sol`

Description: The deployment scripts `DeployNillionImpl.s.sol` and `UpgradeNillion.s.sol` call `vm.envUint("PRIVATE_KEY")`, so the deployer key must live unencrypted in shell environment variables. Environment storage is easily leaked via logs, shell history, or process inspection, exposing signer credentials. This risk affects every deployment that relies on these scripts.

Recommendation: Move signer keys to an encrypted keystore or hardware signer supported by Foundry (see https://getfoundry.sh/guides/best-practices/key-management) and update the scripts to unlock keys securely instead of reading raw environment variables.

NIL-2 Centralized Power

• Low Unresolved

File(s) affected: `NillionBase.sol`, `UpgradeNillion.s.sol`

Description: The `initialize()` function grants `DEFAULT_ADMIN_ROLE`, `MINTER_ROLE`, and `UPGRADER_ROLE` to the deployer, and `UpgradeNillion.s.sol` assigns both `MINTER_ROLE` and `BURNER_ROLE` to the same bridge address. If no multisig is in use, a single compromised key would be able to mint arbitrarily, burn arbitrarily, or upgrade the implementation because all critical roles remain concentrated under the same signer. This centralization pattern must be understood and accepted operationally.

Recommendation: If a single signer currently holds these roles, migrate them to a multisig, set explicit mint caps aligned with supply plans, rotate keys on a defined schedule, and keep burner/minter responsibilities on distinct addresses depending on operational needs.

NIL-3 Missing Storage Gap Introduces Storage Collision Possibility

• Low Unresolved

File(s) affected: `NillionBase.sol`

Description: The `NillionBase` contract currently keeps only constant identifiers, so no storage slots are occupied today. However, because the contract is upgradeable and lacks a reserved `__gap`, any future version that adds state variables could collide with storage written by earlier versions or storage of the `Nillion` contract. As soon as storage variables (anchored at storage slot 0) are implemented in the `Nillion` contract, adding a gap in `NillionBase` would no longer be possible.

Recommendation: Add a fixed-size `uint256[50] private __gap;` (or similar) at the bottom of the storage layout of the `NillionBase` contract, or adopt a EIP-7201 namespaced storage layout. Maintain the gap across versions and adjust its size whenever new variables are appended.

NIL-4 Unsafe Access Control Not Following the Two-Step Pattern

• Low Unresolved

File(s) affected: `NillionBase.sol`, `UpgradeNillion.s.sol`

Description: The `initialize()` function grants `DEFAULT_ADMIN_ROLE` to the deployer, and `UpgradeNillion.s.sol` relies on that address to manage upgrades and roles. Using `AccessControlUpgradeable` means an admin can swap itself out in a single transaction; a typo or compromised signer instantly loses control. OpenZeppelin provides `AccessControlDefaultAdminRulesUpgradeable`, which enforces a two-step transfer with a delay.

Recommendation: Replace the direct admin assignment with `AccessControlDefaultAdminRulesUpgradeable` (or an equivalent two-step flow) so new admins accept their role before it takes effect, or document and accept the operational risk.

NIL-5
Unusual Use of the `initializer` Modifier Could Leave Implementation Partially Unlocked

• Low Unresolved

File(s) affected: `NillionBase.sol`, `Nillion.sol`

Description: The `NillionBase` contract calls the `initializer` modifier inside its constructor. When the implementation is deployed directly, this bumps `_initialized` to `1`, yet `_disableInitializers()` is never invoked (which would set `_initialized = type(uint64).max`). Future upgrades that add `reinitializer(2)` or similar steps would remain callable on the implementation contract, allowing misconfiguration or in the worst-case take over of the implementation contract. However, this issue does not actually occur in practice, since the inheriting `Nillion` contract calls `_disableInitializers()` in its constructor, preventing any further initialization.

Recommendation: Mark `NillionBase` as abstract and remove the constructor, only letting the `Nillion` contract call `_disableInitializers()` in its own constructor. Alternatively, keep the constructor but invoke `_disableInitializers()` instead of `initializer`.

NIL-6 Outdated Solidity Version

• Informational Unresolved

File(s) affected: `NillionBase.sol`, `Nillion.sol`

Description: The contracts are deployed with `pragma solidity ^0.8.4`. Because the pragma uses a caret, different environments may compile with different patch versions, and 0.8.4 no longer reflects the most recent security and optimizer fixes.

Recommendation: Pin the contracts to an exact, currently supported compiler (for example `pragma solidity 0.8.30`) and document the choice.

NIL-7 No Test Suite

• Informational Unresolved

Description: The repository contains no unit or integration tests, so core behaviors such as role restrictions, mint/burn limits, upgrade workflows —are never tested. Even simple ERC-20 contracts benefit from tests to catch bugs during upgrades.

Recommendation: Set up a minimal Foundry test suite covering initialization, role-based mint/burn paths, and upgrade authorization. Expand it as new features ship so production changes remain verifiable.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not pose an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition

or potential vulnerability that was not within the scope of the review.

Files

Repo: `https://github.com/NillionNetwork/nil-erc20-token`

- `328...bef ./script/DeployNillionImpl.s.sol`
- `93d...aa5 ./script/UpgradeNillion.s.sol`
- `6b4...fab ./src/Nillion/contracts/Nillion.sol`
- `53b...63b ./src/Nillion/contracts/NillionBase.sol`

Test Suite Results

No test suite.

Code Coverage

No test suite.

Changelog

- 2025-10-21 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.