Ether Authority

etherauthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Degen Token
Website:	degen.tips
Platform:	Base Chain Network
Language:	Solidity
Date:	June 11th, 2024

Table of contents

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Degen Token smart contract from degen.tips was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on June 11th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

Website Details

Degen, an ERC-20 token launched in January 2024, has reshaped the Farcaster ecosystem by enabling Casters to reward others with DEGEN for posting quality content.

Our points system recognizes unique posts and comments, effectively bridging the gap between online contributions and real-world value.

Degen.tips focuses on the Degen token, an ERC-20 token integrated with the Farcaster ecosystem. It enables users to reward quality content with DEGEN tokens, turning online engagement into tangible value.

The platform supports various activities including airdrops, bounties, and development on its EVM-compatible blockchain, Degen L3, which features low transaction fees.

Code Details

- The `DegenToken` contract is a custom ERC-20 token with burnable, pausable, and votable features, utilizing OpenZeppelin libraries. Key features include:
  - **Token Details:** Named "Degen" with the symbol "DEGEN" and an initial supply of 36,965,935,954 tokens.
  - **Minting Restrictions:** Minting can only occur once a year, capped at 1% of the total supply.
  - **Ownership:** Only the contract owner can mint, pause, or unpause the token.
  - **Security:** Includes various error checks for minting conditions.
- This ensures a secure and controlled token issuance process.

Audit scope

Name	**Code Review and Security Analysis Report for Degen Token Smart Contract**
Platform	**Base Chain Network**
Language	**Solidity**
File	**DegenToken.sol**
Smart Contract Code	0x4ed4e862860bed51a9570b96d89af5e1b0efefed
Audit Date	June 11th,2024
Audit Result	**Passed**

Code Audit History

1	0	0	0	0	1
Total Findings	Critical	High	Medium	Low	Informational

Severity Definitions

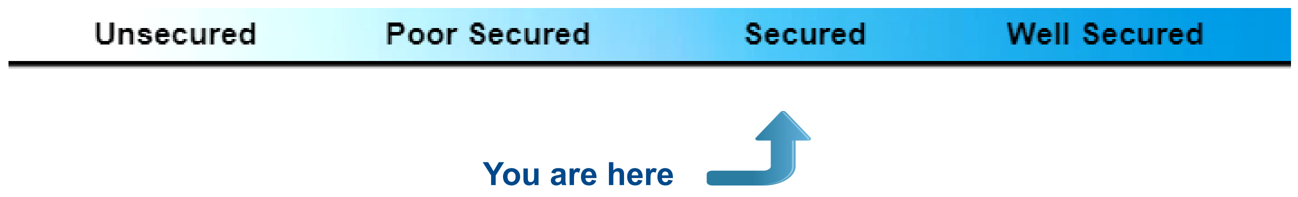
0 🟥	**Critical**		Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
0 🟧	**High**		High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. Public access is crucial.
0 🟨	**Medium**		Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
0 🟩	**Low**		Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
1 🟦	**Lowest / Informational / Best Practice**	🟦	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Degen • Symbol: DEGEN • Decimals: 18	**YES, This is valid.**
Owner Specifications: • Mint new tokens. • Pause/Unpause all token transfers. • Renounce the ownership of the contract to leave the contract without an owner. • Transfers ownership of the contract to a new account.	**YES, This is valid.** **We advise renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **"Secured"**.Also, these contracts contain owner control, which does not make them fully decentralized.



We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low, and 1 very low-level issue.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Moderated
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

Business Risk Analysis

Category	Result
● Buy Tax	0%
● Sell Tax	0%
● Cannot Buy	No
● Cannot Sell	No
● Max Tax	0%
● Modify Tax	No
● Fee Check	Not Detected
● Is Honeypot	Not Detected
● Trading Cooldown	Not Detected
● Can Pause Trade?	Not Detected
● Pause Transfer?	Yes
● Max Tax?	No
● Is it Anti-whale?	Not Detected
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	No
● Blacklist Check	No
● Can Mint?	Yes
● Is it a Proxy Contract?	No
● Is it used Open Source?	Yes
● External Call Risk?	No
● Balance Modifiable?	No
● Can Take Ownership?	Yes
● Ownership Renounce?	Yes
● Hidden Owner?	Not Detected
● Self Destruction?	Not Detected
● Auditor Confidence	High

Overall Audit Result: PASSED

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Degen Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Degen Token.

The EtherAuthority team has not provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Degen Token smart contract code in the form of a basescan web link.

As mentioned above, code parts are well commented on. And the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

DegenToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	mint	external	access only Owner	No Issue
3	pause	write	Gas Optimization	Refer Audit Findings
4	unpause	write	Gas Optimization	Refer Audit Findings
5	_update	internal	Passed	No Issue
6	nonces	read	Gas Optimization	Refer Audit Findings
7	name	read	Passed	No Issue
8	symbol	read	Passed	No Issue
9	decimals	read	Passed	No Issue
10	totalSupply	read	Passed	No Issue
11	balanceOf	read	Passed	No Issue
12	transfer	write	Passed	No Issue
13	allowance	read	Passed	No Issue
14	approve	write	Passed	No Issue
15	transferFrom	write	Passed	No Issue
16	_transfer	internal	Passed	No Issue
17	_update	internal	Passed	No Issue
18	_mint	internal	Passed	No Issue
19	_burn	internal	Passed	No Issue
20	_approve	internal	Passed	No Issue
21	_approve	internal	Passed	No Issue
22	_spendAllowance	internal	Passed	No Issue
23	burn	write	Passed	No Issue
24	burnFrom	write	Passed	No Issue
25	_update	internal	Passed	No Issue
26	permit	write	Passed	No Issue
27	nonces	read	Passed	No Issue
28	DOMAIN_SEPARATOR	external	Passed	No Issue
29	clock	read	Passed	No Issue
30	CLOCK_MODE	read	Passed	No Issue
31	getVotes	read	Passed	No Issue
32	getPastVotes	read	Passed	No Issue
33	getPastTotalSupply	read	Passed	No Issue
34	_getTotalSupply	internal	Passed	No Issue
35	delegates	read	Passed	No Issue
36	delegate	write	Passed	No Issue
37	delegateBySig	write	Passed	No Issue

38	_delegate	internal	Passed	No Issue
39	_transferVotingUnits	internal	Passed	No Issue
40	_moveDelegateVotes	write	Passed	No Issue
41	_numCheckpoints	internal	Passed	No Issue
42	_checkpoints	internal	Passed	No Issue
43	_push	write	Passed	No Issue
44	_add	write	Passed	No Issue
45	_subtract	write	Passed	No Issue
46	_getVotingUnits	internal	Passed	No Issue
47	_maxSupply	internal	Passed	No Issue
48	_update	internal	Passed	No Issue
49	_getVotingUnits	internal	Passed	No Issue
50	numCheckpoints	read	Passed	No Issue
51	checkpoints	read	Passed	No Issue
52	onlyOwner	modifier	Passed	No Issue
53	owner	read	Passed	No Issue
54	_checkOwner	internal	Passed	No Issue
55	renounceOwnership	write	access only Owner	No Issue
56	transferOwnership	write	access only Owner	No Issue
57	_transferOwnership	internal	Passed	No Issue

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

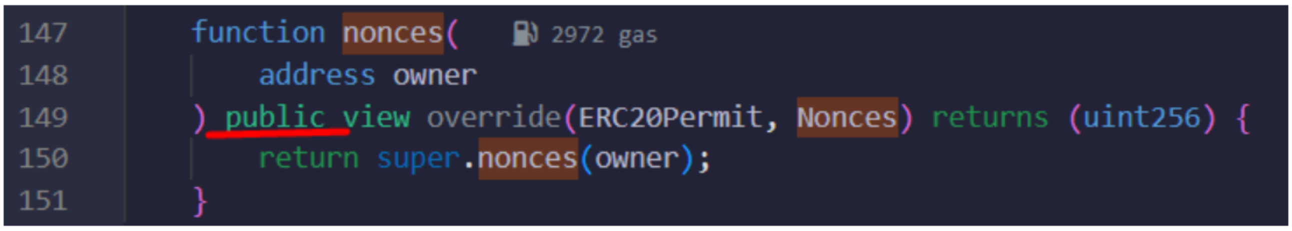
No Medium-severity vulnerabilities were found.

Low

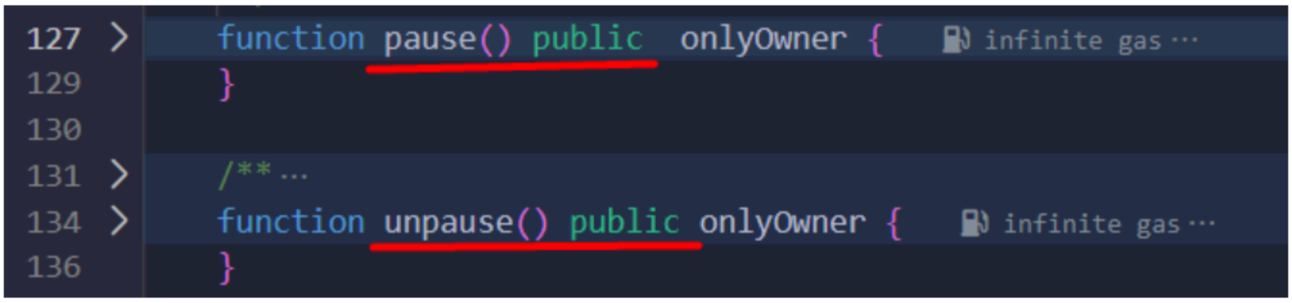
No Low-severity vulnerabilities were found.

Very Low / Informational / Best practices:

[I-01] Gas Optimization:



```
    function nonces(    2972 gas
        address owner
    ) public view override(ERC20Permit, Nonces) returns (uint256) {
        return super.nonces(owner);
    }
```



```
    function pause() public  onlyOwner {    infinite gas ···
    }

    /** ...
    function unpause() public onlyOwner {    infinite gas ···
    }
```

Description:

These 'public' functions that are never called by the contract could be declared external functions.

Recommendation: We suggest declaring these functions external for better Gas optimization.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble.

Centralized
Decentralized
You are here

The following are Admin functions:

DegenToken.sol

- mint: Mint new tokens only by the owner.
- pause: Pause all token transfers only by the owner.
- unpause: Unpause all token transfers only by the owner.

Ownable.sol

- transferOwnership: Allows the current owner to transfer control of the contract to a newOwner.
- renounceOwnership(): Renounce the ownership of the contract to leave the contract without an owner.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of a basescan web link. We have used all possible tests based on given objects as files. We observed 1 Informational issue in the smart contracts. but those are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:
In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:
Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

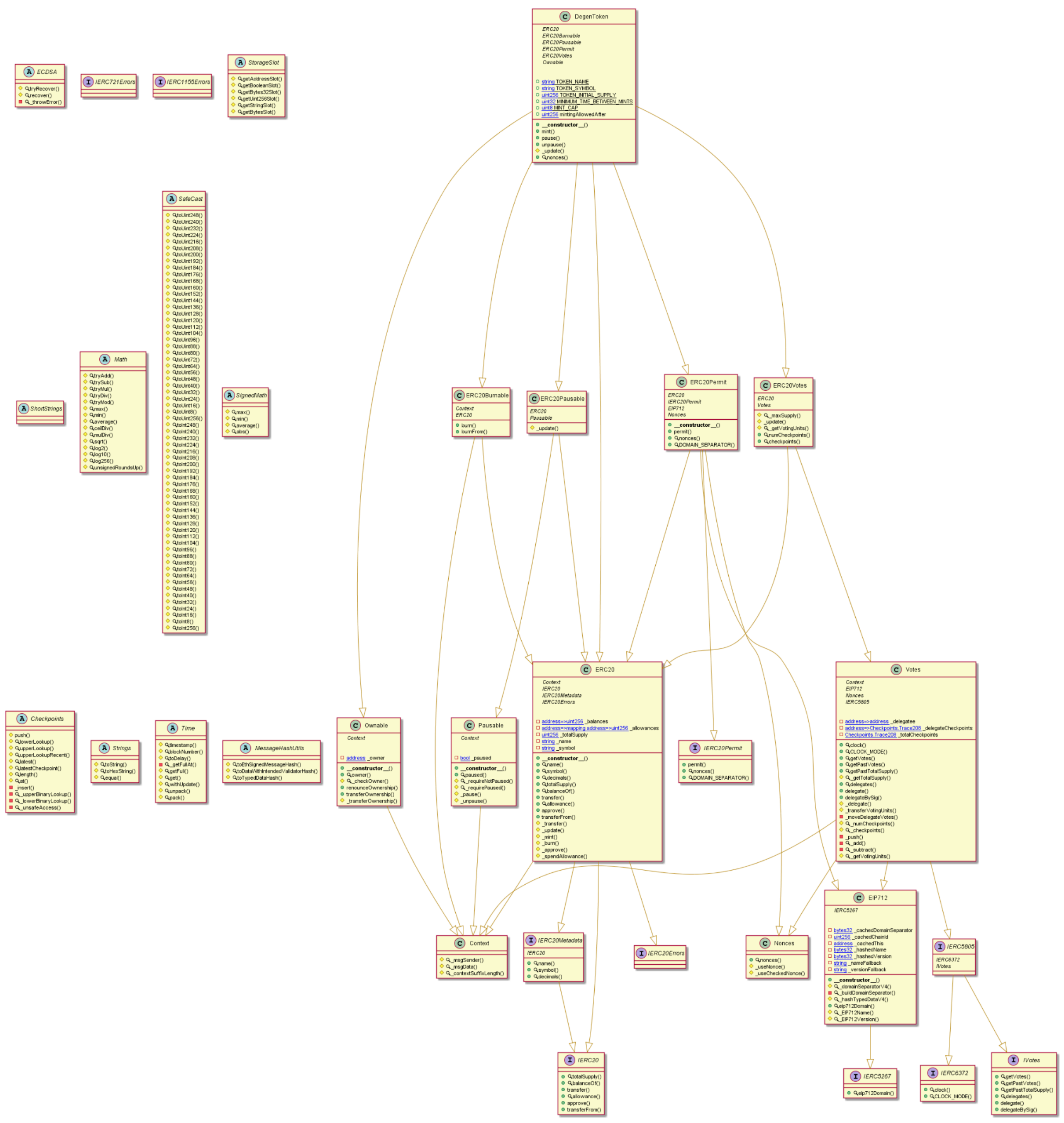
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Degen Token



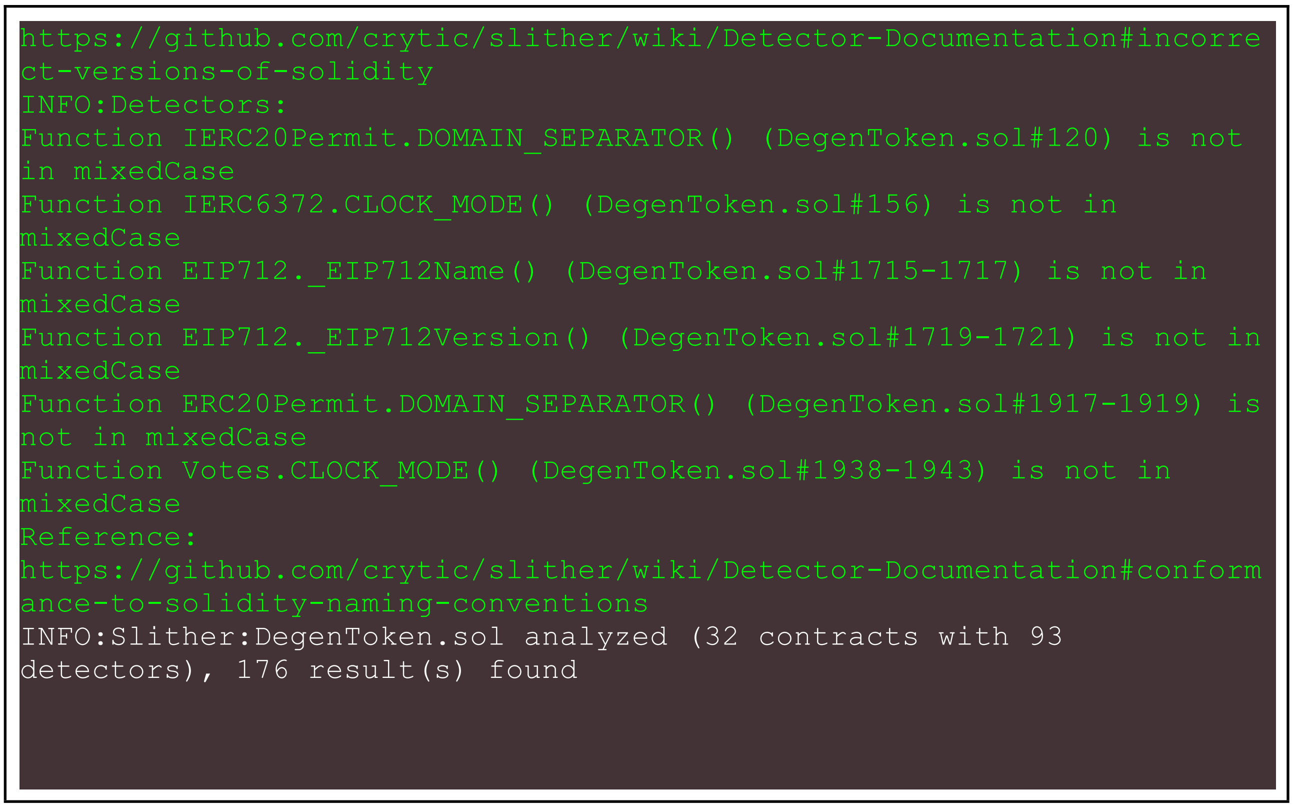
Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

DegenToken.sol

```
INFO:Detectors:
DegenToken.nonces(address).owner (DegenToken.sol#2173) shadows:
        - Ownable.owner() (DegenToken.sol#1549-1551) (function)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#local-v
ariable-shadowing
INFO:Detectors:
Votes.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32)
(DegenToken.sol#1978-1997) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > expiry (DegenToken.sol#1986)
DegenToken.constructor(uint256) (DegenToken.sol#2123-2136) uses
timestamp for comparisons
        Dangerous comparisons:
        - mintingAllowedAfter_ < block.timestamp
(DegenToken.sol#2130)
DegenToken.mint(address,uint96) (DegenToken.sol#2138-2154) uses
timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < mintingAllowedAfter (DegenToken.sol#2139)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#block-t
imestamp
INFO:Detectors:
Votes._add(uint208,uint208) (DegenToken.sol#2057-2059) is never used
and should be removed
Votes._getTotalSupply() (DegenToken.sol#1965-1967) is never used and
should be removed
Votes._subtract(uint208,uint208) (DegenToken.sol#2061-2063) is never
used and should be removed
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#dead-co
de
INFO:Detectors:
Pragma version^0.8.20 (DegenToken.sol#3) necessitates a version too
recent to be trusted. Consider deploying with 0.8.18.
solc-0.8.20 is not recommended for deployment
Reference:
```

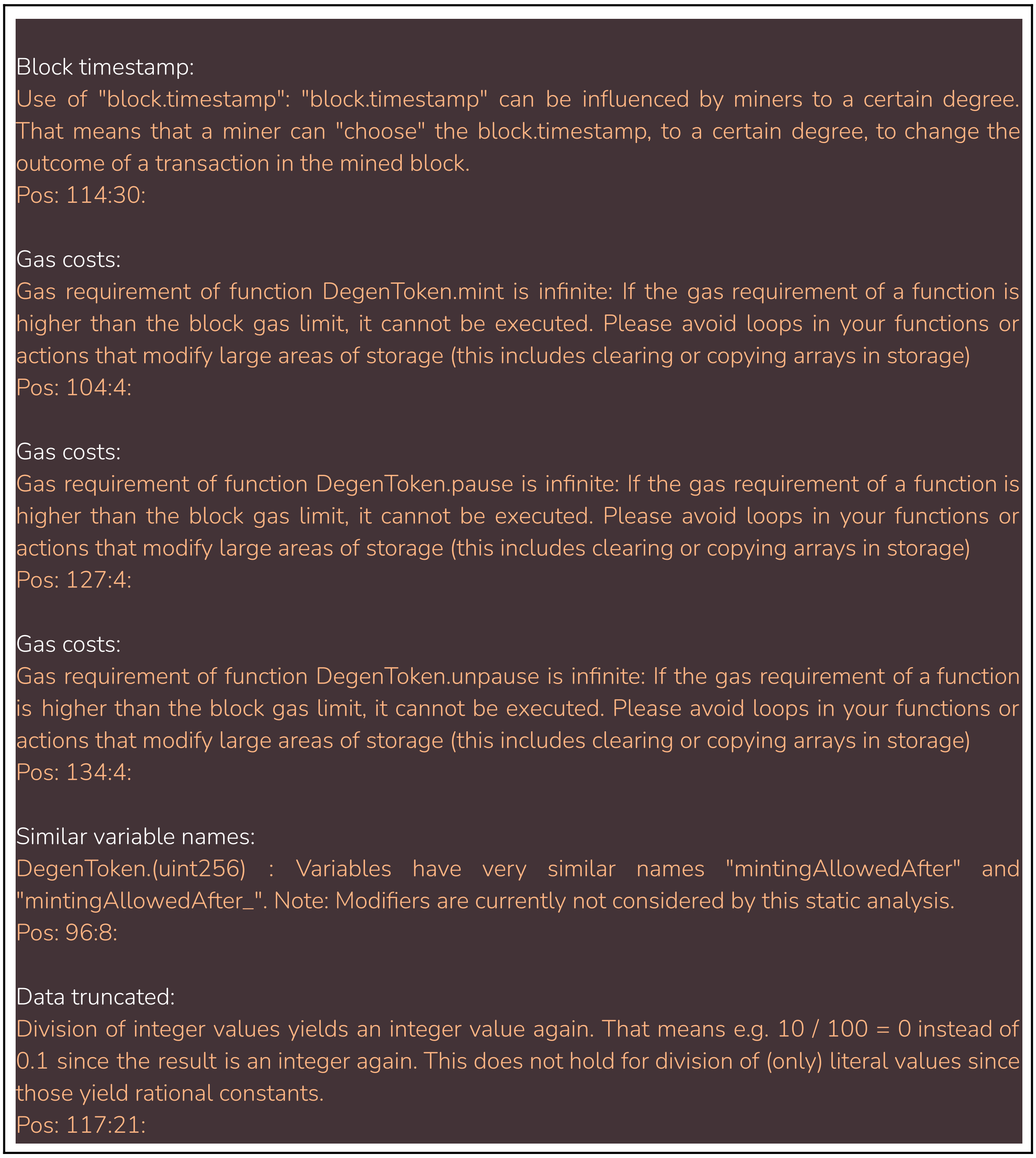


```
https://github.com/crytic/slither/wiki/Detector-Documentation#incorre
ct-versions-of-solidity
INFO:Detectors:
Function IERC20Permit.DOMAIN_SEPARATOR() (DegenToken.sol#120) is not
in mixedCase
Function IERC6372.CLOCK_MODE() (DegenToken.sol#156) is not in
mixedCase
Function EIP712._EIP712Name() (DegenToken.sol#1715-1717) is not in
mixedCase
Function EIP712._EIP712Version() (DegenToken.sol#1719-1721) is not in
mixedCase
Function ERC20Permit.DOMAIN_SEPARATOR() (DegenToken.sol#1917-1919) is
not in mixedCase
Function Votes.CLOCK_MODE() (DegenToken.sol#1938-1943) is not in
mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conform
ance-to-solidity-naming-conventions
INFO:Slither:DegenToken.sol analyzed (32 contracts with 93
detectors), 176 result(s) found
```

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

DegenToken.sol

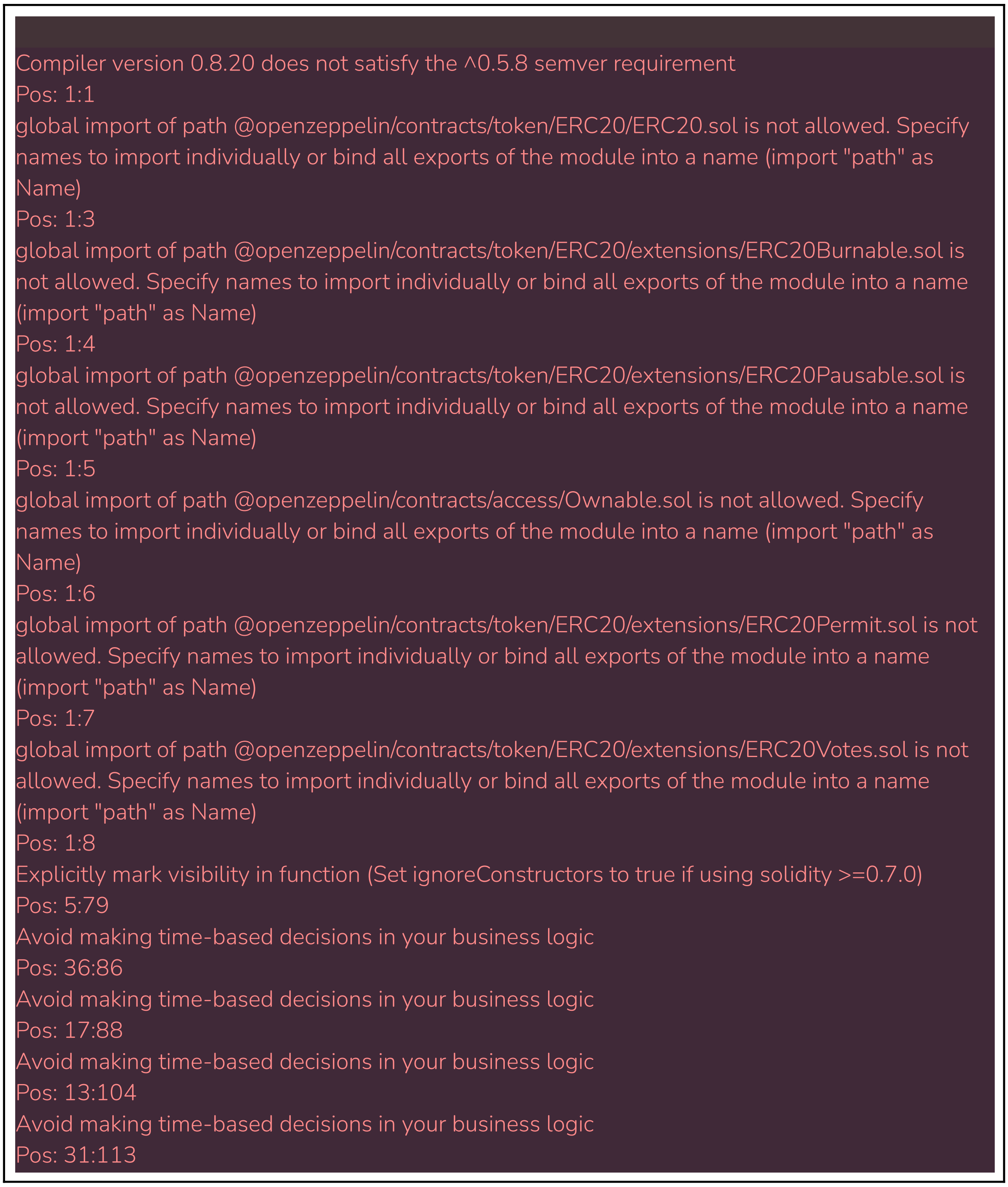


```
Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.
That means that a miner can "choose" the block.timestamp, to a certain degree, to change the
outcome of a transaction in the mined block.
Pos: 114:30:

Gas costs:
Gas requirement of function DegenToken.mint is infinite: If the gas requirement of a function is
higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or
actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 104:4:

Gas costs:
Gas requirement of function DegenToken.pause is infinite: If the gas requirement of a function is
higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or
actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 127:4:

Gas costs:
Gas requirement of function DegenToken.unpause is infinite: If the gas requirement of a function
is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or
actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 134:4:

Similar variable names:
DegenToken.(uint256) : Variables have very similar names "mintingAllowedAfter" and
"mintingAllowedAfter_". Note: Modifiers are currently not considered by this static analysis.
Pos: 96:8:

Data truncated:
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of
0.1 since the result is an integer again. This does not hold for division of (only) literal values since
those yield rational constants.
Pos: 117:21:
```

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

DegenToken.sol



```
Compiler version 0.8.20 does not satisfy the ^0.5.8 semver requirement
Pos: 1:1
global import of path @openzeppelin/contracts/token/ERC20/ERC20.sol is not allowed. Specify
names to import individually or bind all exports of the module into a name (import "path" as
Name)
Pos: 1:3
global import of path @openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol is
not allowed. Specify names to import individually or bind all exports of the module into a name
(import "path" as Name)
Pos: 1:4
global import of path @openzeppelin/contracts/token/ERC20/extensions/ERC20Pausable.sol is
not allowed. Specify names to import individually or bind all exports of the module into a name
(import "path" as Name)
Pos: 1:5
global import of path @openzeppelin/contracts/access/Ownable.sol is not allowed. Specify
names to import individually or bind all exports of the module into a name (import "path" as
Name)
Pos: 1:6
global import of path @openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol is not
allowed. Specify names to import individually or bind all exports of the module into a name
(import "path" as Name)
Pos: 1:7
global import of path @openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol is not
allowed. Specify names to import individually or bind all exports of the module into a name
(import "path" as Name)
Pos: 1:8
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:79
Avoid making time-based decisions in your business logic
Pos: 36:86
Avoid making time-based decisions in your business logic
Pos: 17:88
Avoid making time-based decisions in your business logic
Pos: 13:104
Avoid making time-based decisions in your business logic
Pos: 31:113
```

Software analysis result:

This software reported many false positive results and some were informational issues. So, those issues can be safely ignored.