EtherAuthority

etherauthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Wormhole Token
Website:	w-token
Platform:	Base Chain Network
Language:	Solidity
Date:	May 29th, 2024

Table of contents

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Wormhole Token smart contract from wormhole.com/ecosystem/w-token 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 May 29th, 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

- W is the native token that powers the Wormhole platform.
- The W token powers the Wormhole platform and is designed for multichain use.
- It initially launched as a Solana SPL token, with ERC20 functionality to be added later through Wormhole's Native Token Transfers.
- The W token has a max supply of 10 billion and an initial circulating supply of 1.8 billion. 82% of the tokens are initially locked and will unlock over four years according to a vesting schedule.
- W will initially launch as a native Solana SPL token. ERC20 functionality will be enabled post-launch through Wormhole's Native Token Transfers (NTT), allowing seamless transfers across any Wormhole-connected network.

Code Details

- This Solidity contract, `WToken`, is an upgradeable governance token with controlled minting and burning capabilities, utilizing the OpenZeppelin library for various functionalities. Below, break down the main components and their functionalities:
- **Key Features and Components:**
  - **Token Minting and Burning:**
    - **mint:** Allows minting of new tokens, with auto-delegation if the recipient doesn't have a delegate.
    - **burn:** Allows burning of the caller's tokens, restricted by `BURNER_ROLE`.
    - **burnFrom:** Not implemented to prevent unintended use.
  - **Governance and Voting:**
    - Uses a timestamp-based clock for governance functions (`clock`, `CLOCK_MODE`).
    - maxSupply: Sets the maximum token supply.
    - setDelegate: Allows setting delegate votes for an account, controlled by `SET_DELEGATE_ROLE`.
- This contract combines the functionality of a traditional ERC20 token with enhanced features for minting, burning, and governance, all while ensuring secure, role-based access control and the capability to upgrade the contract as needed. The use of OpenZeppelin libraries provides robust, audited code, enhancing security and reliability.

Audit scope

Name	**Code Review and Security Analysis Report for Wormhole Token Smart Contract**
Platform	**Base Chain Network**
Language	**Solidity**
File	WToken.sol
Smart Contract Code	0x344518934533ec82b49ea533b196dce5cfa64411
Audit Date	May 29th,2024
Audit Result	**Passed**

Code Audit History

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

Severity Definitions

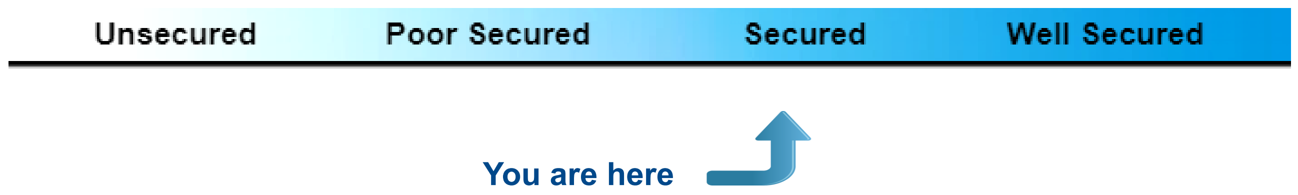
0 🟥	**Critical**		Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
0 🟧	**High**		High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. Public access is crucial.
0 🟨	**Medium**		Medium-level 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
Key Features: • **Token Minting and Burning:** ○ mint: Allows minting new tokens to a specified account, with auto-delegation if the recipient doesn't have a delegate. ○ burn: Allows burning of tokens held by the caller, restricted to addresses with the \`BURNER_ROLE\`. ○ burnFrom: Not implemented to prevent unintended use. • **Governance and Voting:** ○ Implements a timestamp-based clock for governance functions (\`clock\`, \`CLOCK_MODE\`). ○ \`_maxSupply\`: Sets the maximum token supply. ○ \`setDelegate\`: Allows setting a delegate for an account, controlled by the \`SET_DELEGATE_ROLE\`.	**YES, This is valid.**

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	Moderated
Gas Optimization	“Out of Gas” Issue	Passed
	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?	No
● Max Tax?	No
● Is it Anti-whale?	Not Detected
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	No
● Blacklist Check	No
● Can Mint?	No
● Is it a Proxy Contract?	Yes
● Is it used Open Source?	Yes
● External Call Risk?	No
● Balance Modifiable?	No
● Can Take Ownership?	No
● Ownership Renounce?	No
● 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 Wormhole 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 Wormhole 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 Wormhole 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 used in external smart contract calls.

AS-IS overview

WToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initialize	external	initializer	No Issue
3	grantMinterAndBurner	external	Passed	No Issue
4	clock	read	Passed	No Issue
5	CLOCK_MODE	read	Passed	No Issue
6	burn	write	Passed	No Issue
7	burnFrom	write	Passed	No Issue
8	mint	external	Passed	No Issue
9	setDelegate	external	Passed	No Issue
10	_maxSupply	internal	Passed	No Issue
11	nonces	read	Passed	No Issue
12	_update	internal	Passed	No Issue
13	_authorizeUpgrade	internal	Unused function	Refer Audit Findings
14	_getAccessControlDefaultAdminRulesStorage	write	Passed	No Issue
15	__AccessControlDefaultAdminRules_init	internal	access only Initializing	No Issue
16	__AccessControlDefaultAdminRules_init_unchained	internal	access only Initializing	No Issue
17	supportsInterface	read	Passed	No Issue
18	owner	read	Passed	No Issue
19	grantRole	write	Passed	No Issue
20	revokeRole	write	Passed	No Issue
21	renounceRole	write	Passed	No Issue
22	_grantRole	internal	Passed	No Issue
23	_revokeRole	internal	Passed	No Issue
24	_setRoleAdmin	internal	Passed	No Issue
25	defaultAdmin	read	Passed	No Issue
26	pendingDefaultAdmin	read	Passed	No Issue
27	defaultAdminDelay	read	Passed	No Issue
28	pendingDefaultAdminDelay	read	Passed	No Issue
29	defaultAdminDelayIncreaseWait	read	Passed	No Issue
30	beginDefaultAdminTransfer	write	default Admin Role	No Issue
31	_beginDefaultAdminTransfer	internal	Passed	No Issue
32	cancelDefaultAdminTransfer	write	default Admin Role	No Issue
33	_cancelDefaultAdminTransfer	internal	Passed	No Issue
34	acceptDefaultAdminTransfer	write	Passed	No Issue
35	_acceptDefaultAdminTransfer	internal	Passed	No Issue

36	changeDefaultAdminDelay	write	default Admin Role	No Issue
37	_changeDefaultAdminDelay	internal	Passed	No Issue
38	rollbackDefaultAdminDelay	write	default Admin Role	No Issue
39	_rollbackDefaultAdminDelay	internal	Passed	No Issue
40	_delayChangeWait	internal	Passed	No Issue
41	_setPendingDefaultAdmin	write	Passed	No Issue
42	_setPendingDelay	write	Passed	No Issue
43	_isScheduleSet	write	Passed	No Issue
44	_hasSchedulePassed	read	Passed	No Issue
45	__ERC20Burnable_init	internal	access only Initializing	No Issue
46	__ERC20Burnable_init_unchained	internal	access only Initializing	No Issue
47	burn	write	Passed	No Issue
48	burnFrom	write	Passed	No Issue
49	__ERC20Permit_init	internal	access only Initializing	No Issue
50	__ERC20Permit_init_unchained	internal	access only Initializing	No Issue
51	permit	write	Passed	No Issue
52	nonces	read	Passed	No Issue
53	DOMAIN_SEPARATOR	external	Passed	No Issue
54	__ERC20Votes_init	internal	access only Initializing	No Issue
55	__ERC20Votes_init_unchained	internal	access only Initializing	No Issue
56	_maxSupply	internal	Passed	No Issue
57	_update	internal	Passed	No Issue
58	_getVotingUnits	internal	Passed	No Issue
59	numCheckpoints	read	Passed	No Issue
60	checkpoints	read	Passed	No Issue
61	_getERC20Storage	write	Passed	No Issue
62	__ERC20_init	internal	access only Initializing	No Issue
63	__ERC20_init_unchained	internal	access only Initializing	No Issue
64	name	read	Passed	No Issue
65	symbol	read	Passed	No Issue
66	decimals	read	Passed	No Issue
67	totalSupply	read	Passed	No Issue
68	balanceOf	read	Passed	No Issue
69	transfer	write	Passed	No Issue
70	allowance	read	Passed	No Issue
71	approve	write	Passed	No Issue
72	transferFrom	write	Passed	No Issue
73	_transfer	internal	Passed	No Issue

74	_update	internal	Passed	No Issue
75	_mint	internal	Passed	No Issue
76	_burn	internal	Passed	No Issue
77	_approve	internal	Passed	No Issue
78	_approve	internal	Passed	No Issue
79	_spendAllowance	internal	Passed	No Issue
80	onlyProxy	modifier	Passed	No Issue
81	notDelegated	modifier	Passed	No Issue
82	__UUPSUpgradeable_init	internal	access only Initializing	No Issue
83	__UUPSUpgradeable_init_unchained	internal	access only Initializing	No Issue
84	proxiableUUID	external	Passed	No Issue
85	upgradeToAndCall	write	access only Proxy	No Issue
86	_checkProxy	internal	Passed	No Issue
87	_checkNotDelegated	internal	Passed	No Issue
88	_authorizeUpgrade	internal	Passed	No Issue
89	_upgradeToAndCallUUPS	write	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] Unused function:

```
 function _authorizeUpgrade(address /* newImplementation */ )
internal view override {
        _checkRole(DEFAULT_ADMIN_ROLE);
    }
```

Description:

The "_authorizeUpgrade" function is defined in the contract but never used.

Recommendation: We suggest removing this function.

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:

AccessControlDefaultAdminRulesUpgradeable.sol

- grantRole: Grants `role` to `account` can be set by the admin.
- revokeRole: Revokes `role` from `account` by the admin.
- renounceRole: Renounce Role from `account` by the admin.
- _revokeRole: Revokes `role` from `account` by the admin.
- _setRoleAdmin: The current admin can place all roles.
- beginDefaultAdminTransfer: The current admin can transfer the default admin address.
- cancelDefaultAdminTransfer: The current admin can cancel the default admin address.
- changeDefaultAdminDelay: Delay time can be set by default admin.
- rollbackDefaultAdminDelay: Default admin can rollback.

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. And 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).

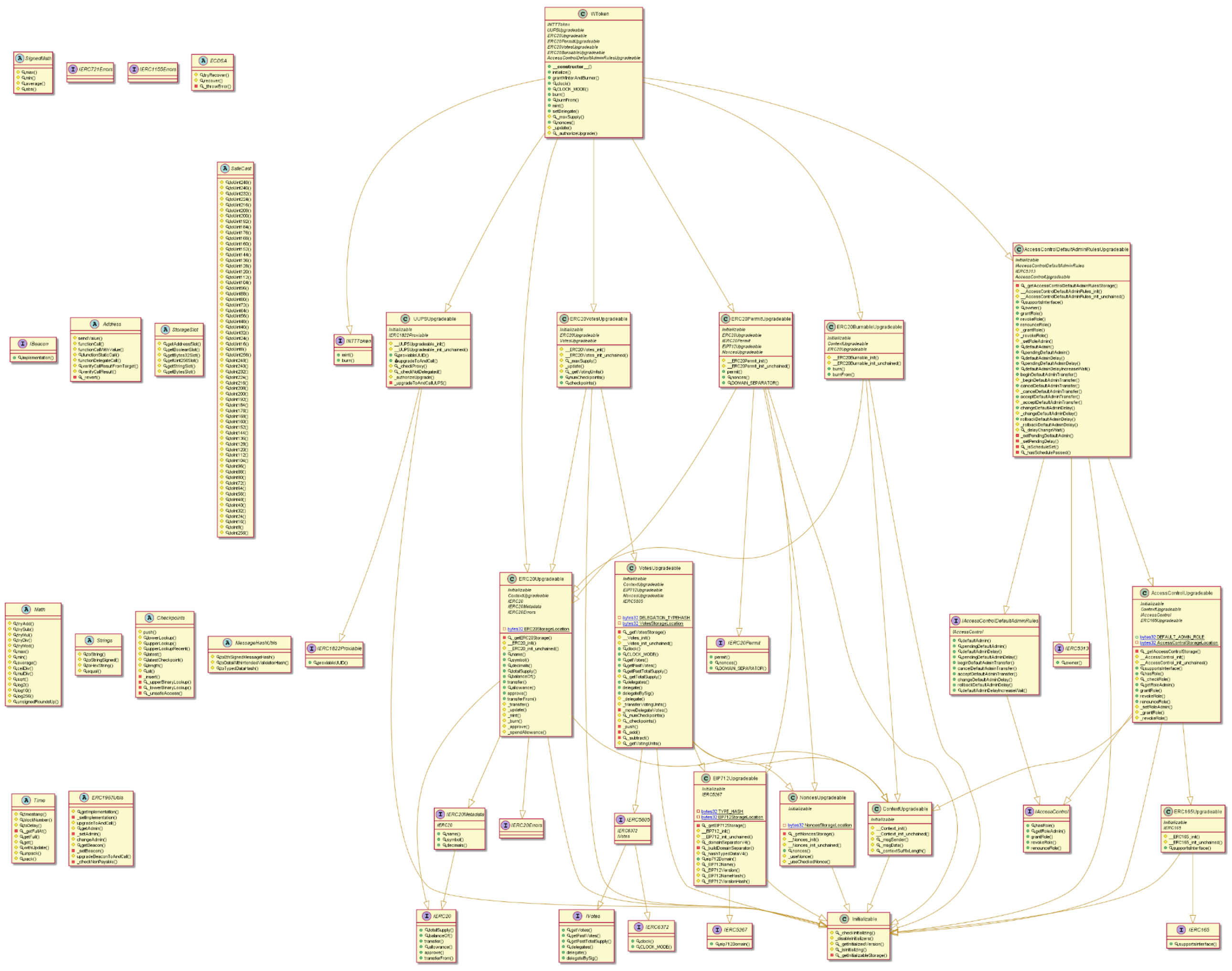
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status or any other statements of the contract. While 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 - Wormhole 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.

WToken.sol

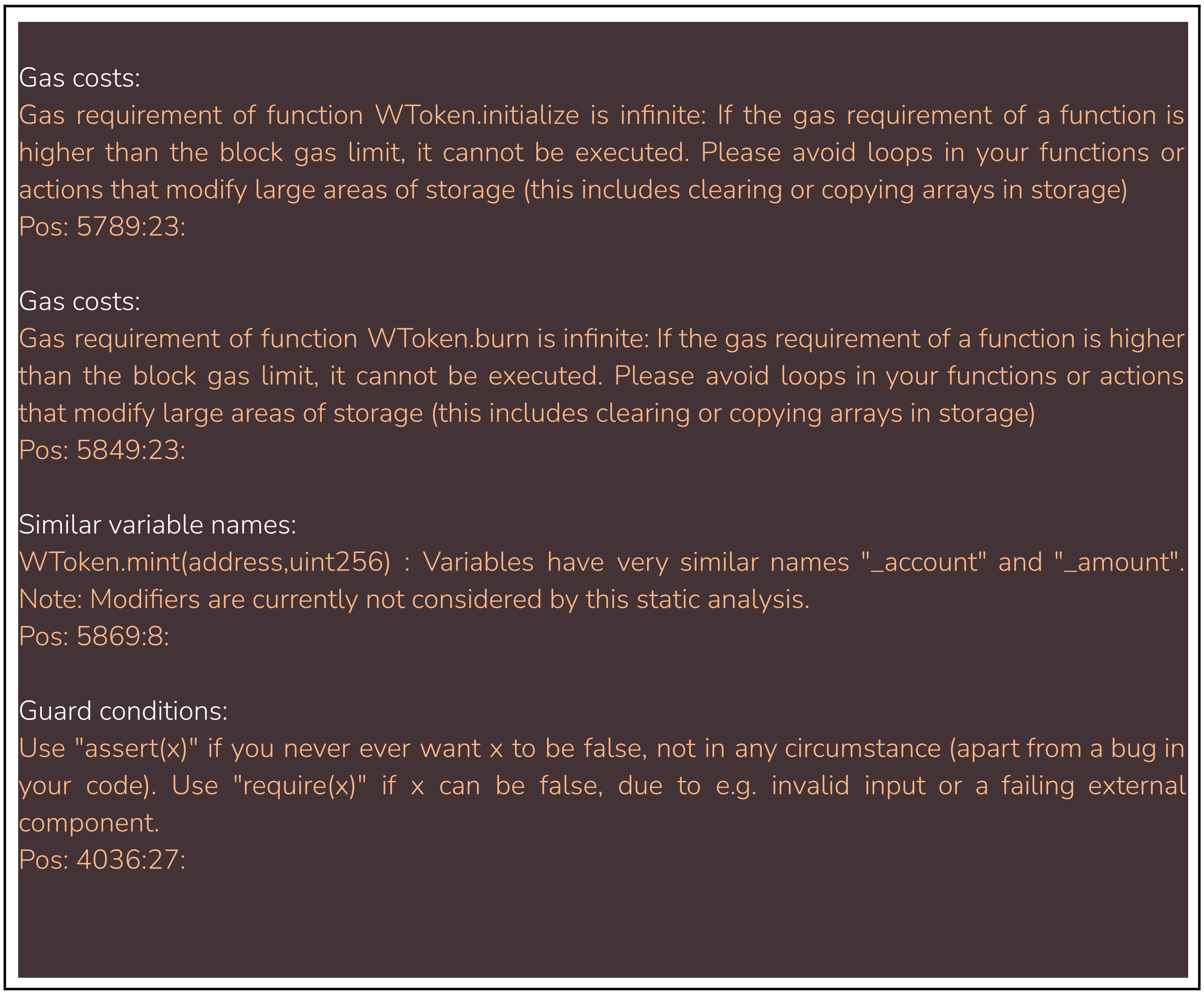
```
INFO:Detectors:
VotesUpgradeable.__Votes_init() (WToken.sol#2152-2153) is never used and should be
removed
VotesUpgradeable.__Votes_init_unchained() (WToken.sol#2155-2156) is never used and should
be removed
VotesUpgradeable._add(uint208,uint208) (WToken.sol#2286-2288) is never used and should
be removed
VotesUpgradeable._getTotalSupply() (WToken.sol#2188-2191) is never used and should be
removed
VotesUpgradeable._subtract(uint208,uint208) (WToken.sol#2290-2292) is never used and
should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.20 (WToken.sol#3) necessitates a version too recent to be trusted. Consider
deploying with 0.8.18.
solc-0.8.25 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter WToken.initialize(string,string,address,address,uint48)._name (WToken.sol#2790) is
not in mixedCase
Parameter WToken.initialize(string,string,address,address,uint48)._symbol (WToken.sol#2791) is
not in mixedCase
Parameter WToken.initialize(string,string,address,address,uint48)._minterAndBurnerAdmin
(WToken.sol#2792) is not in mixedCase
Parameter WToken.initialize(string,string,address,address,uint48)._owner (WToken.sol#2793) is
not in mixedCase
Parameter WToken.initialize(string,string,address,address,uint48)._adminChangeDelay
(WToken.sol#2794) is not in mixedCase
Parameter WToken.grantMinterAndBurner(address)._minterAndBurner (WToken.sol#2805) is
```

```
not in mixedCase
Function WToken.CLOCK_MODE() (WToken.sol#2815-2819) is not in mixedCase
Parameter WToken.burn(uint256)._value (WToken.sol#2821) is not in mixedCase
Parameter WToken.mint(address,uint256)._account (WToken.sol#2829) is not in mixedCase
Parameter WToken.mint(address,uint256)._amount (WToken.sol#2829) is not in mixedCase
Parameter WToken.setDelegate(address,address)._account (WToken.sol#2837) is not in
mixedCase
Parameter WToken.setDelegate(address,address)._delegatee (WToken.sol#2837) is not in
mixedCase
Parameter WToken.nonces(address)._owner (WToken.sol#2846) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Slither:WToken.sol analyzed (42 contracts with 93 detectors), 279 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.

WToken.sol

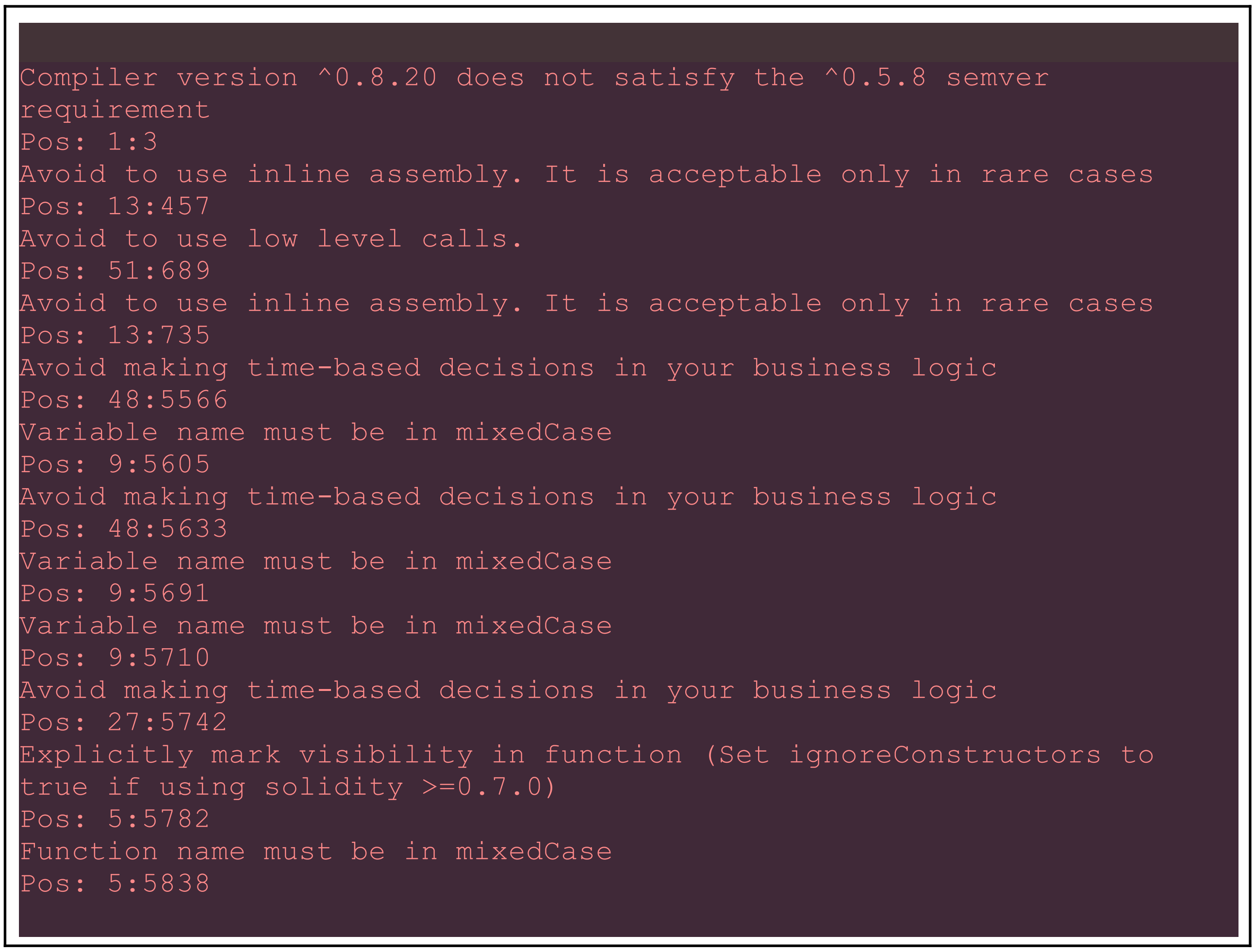


```
Gas costs:
Gas requirement of function WToken.initialize is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 5789:23:

Gas costs:
Gas requirement of function WToken.burn 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: 5849:23:

Similar variable names:
WToken.mint(address,uint256) : Variables have very similar names "_account" and "_amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 5869:8:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 4036:27:
```

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.

WToken.sol



```
Compiler version ^0.8.20 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:3
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 13:457
Avoid to use low level calls.
Pos: 51:689
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 13:735
Avoid making time-based decisions in your business logic
Pos: 48:5566
Variable name must be in mixedCase
Pos: 9:5605
Avoid making time-based decisions in your business logic
Pos: 48:5633
Variable name must be in mixedCase
Pos: 9:5691
Variable name must be in mixedCase
Pos: 9:5710
Avoid making time-based decisions in your business logic
Pos: 27:5742
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:5782
Function name must be in mixedCase
Pos: 5:5838
```

Software analysis result:

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