Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	dYdX Token
Website:	dydx.exchange
Platform:	Ethereum
Language:	Solidity
Date:	February 19th, 2024

Table of contents

Introduction 4

Project Background 4

Audit Scope 4

Claimed Smart Contract Features 5

Audit Summary 6

Technical Quick Stats 7

Business Risk Analysis 8

Code Quality 9

Documentation 9

Use of Dependencies 9

AS-IS overview 10

Severity Definitions 12

Audit Findings 13

Conclusion 19

Our Methodology 20

Disclaimers 22

Appendix

- Code Flow Diagram 23
- Slither Results Log 24
- Solidity static analysis 26
- Solhint Linter 28

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the dYdX Token smart contract from dydx.exchange 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 February 19th, 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

- The dYdX governance token smart contract defines a sophisticated ERC20 token contract with built-in governance features, including delegation of voting and proposition power.
- The `DydxToken` contract ensures secure and restricted transfers, and minting conditions, and allows the owner to manage an allowlist for token transfers during restricted periods.
- The contract leverages the `SafeMath` library for safe arithmetic operations and provides detailed delegation and snapshot mechanisms for tracking governance power.

Audit scope

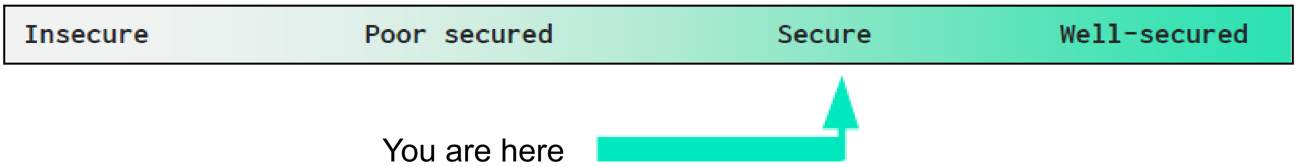
Name	**Code Review and Security Analysis Report for dYdX Token Smart Contract**
Platform	**Ethereum**
Language	**Solidity**
File	DydxToken.sol
Smart Contract Code	0x92d6c1e31e14520e676a687f0a93788b716beff5
Audit Date	February 19th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Name: dYdX ● Symbol: DYDX ● Decimals: 18 ● Total Supply:1 billion	**YES, This is valid.**
Ownership Control: ● Updates addresses to the token transfer allowlist. ● Updates the transfer restriction. ● Mint new tokens. ● Implements the permit function. ● The current owner can transfer the ownership. ● The owner can renounce ownership.	**YES, This is valid.** **We suggest 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, 1 medium, 0 low, and 9 very low level issues.

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	Moderated
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	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage 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	Yes
🟢 Cannot Sell	Yes
🟢 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?	Not Detected
🟢 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?	No
🟢 Can Take Ownership?	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 contract contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in the dYdX 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 dYdX Token.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are not well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a dYdX Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are not well commented on. but 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 are used in this smart contract infrastructure that 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

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Solidity assembly, Compile time warning, Coding style	Refer Audit Findings
2	addToTokenTransferAllowlist	external	Centralization	Refer Audit Findings
3	removeFromTokenTransferAllowlist	external	Centralization	Refer Audit Findings
4	updateTransfersRestrictedBefore	external	Centralization	Refer Audit Findings
5	mint	external	Centralization	Refer Audit Findings
6	permit	external	Passed	No Issue
7	nonces	external	Passed	No Issue
8	transfer	write	Passed	No Issue
9	transferFrom	write	Passed	No Issue
10	_mint	internal	Passed	No Issue
11	_requireTransferAllowed	read	Passed	No Issue
12	_beforeTokenTransfer	internal	Passed	No Issue
13	_getDelegationDataByType	internal	Passed	No Issue
14	delegateByTypeBySig	write	Passed	No Issue
15	delegateBySig	write	Passed	No Issue
16	delegateByType	external	Redundant 'virtual' keyword	Refer Audit Findings
17	delegate	external	Redundant 'virtual' keyword	Refer Audit Findings
18	getDelegateeByType	external	Redundant 'virtual' keyword	Refer Audit Findings
19	getPowerCurrent	external	Redundant 'virtual' keyword	Refer Audit Findings
20	getPowerAtBlock	external	Passed	No Issue
21	_delegateByType	internal	Passed	No Issue
22	_moveDelegatesByType	internal	Passed	No Issue
23	_searchByBlockNumber	internal	Passed	No Issue
24	_getDelegationDataByType	internal	Passed	No Issue
25	_writeSnapshot	internal	Passed	No Issue
26	_getDelegatee	internal	Passed	No Issue
27	owner	read	Passed	No Issue
28	onlyOwner	modifier	Passed	No Issue
29	renounceOwnership	write	Centralization	Refer Audit Findings
30	transferOwnership	write	Centralization	Refer Audit Findings
31	name	read	Passed	No Issue

32	symbol	read	Passed	No Issue
33	decimals	read	Passed	No Issue
34	totalSupply	read	Passed	No Issue
35	balanceOf	read	Passed	No Issue
36	transfer	write	Passed	No Issue
37	allowance	read	Passed	No Issue
38	approve	write	Passed	No Issue
39	transferFrom	write	Passed	No Issue
40	increaseAllowance	write	Passed	No Issue
41	decreaseAllowance	write	Passed	No Issue
42	_transfer	internal	Passed	No Issue
43	_mint	internal	Passed	No Issue
44	_burn	internal	Passed	No Issue
45	_approve	internal	Passed	No Issue
46	_beforeTokenTransfer	internal	Passed	No Issue
47	_setupDecimals	internal	Passed	No Issue
48	_msgSender	internal	Passed	No Issue
49	_msgData	internal	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical Severity

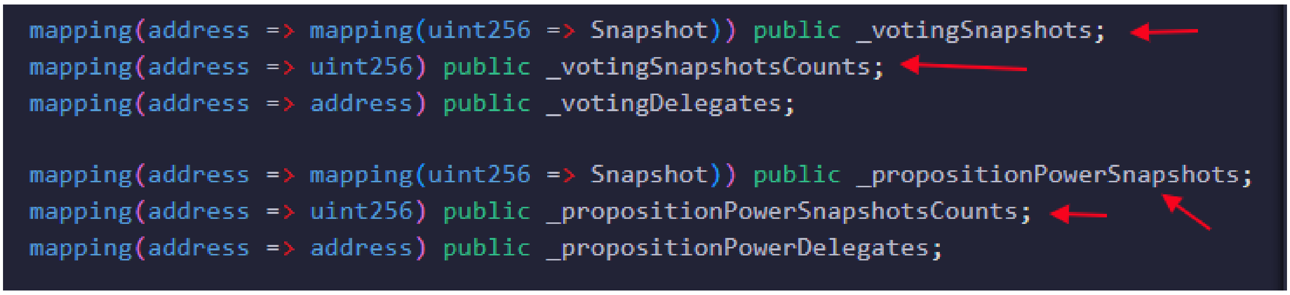
No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

(1) Uninitialized state variable:



```
mapping(address => mapping(uint256 => Snapshot)) public _votingSnapshots;
mapping(address => uint256) public _votingSnapshotsCounts;
mapping(address => address) public _votingDelegates;

mapping(address => mapping(uint256 => Snapshot)) public _propositionPowerSnapshots;
mapping(address => uint256) public _propositionPowerSnapshotsCounts;
mapping(address => address) public _propositionPowerDelegates;
```

Uninitialized state variables.

Resolution: Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

Low

No Low Severity vulnerabilities were found.

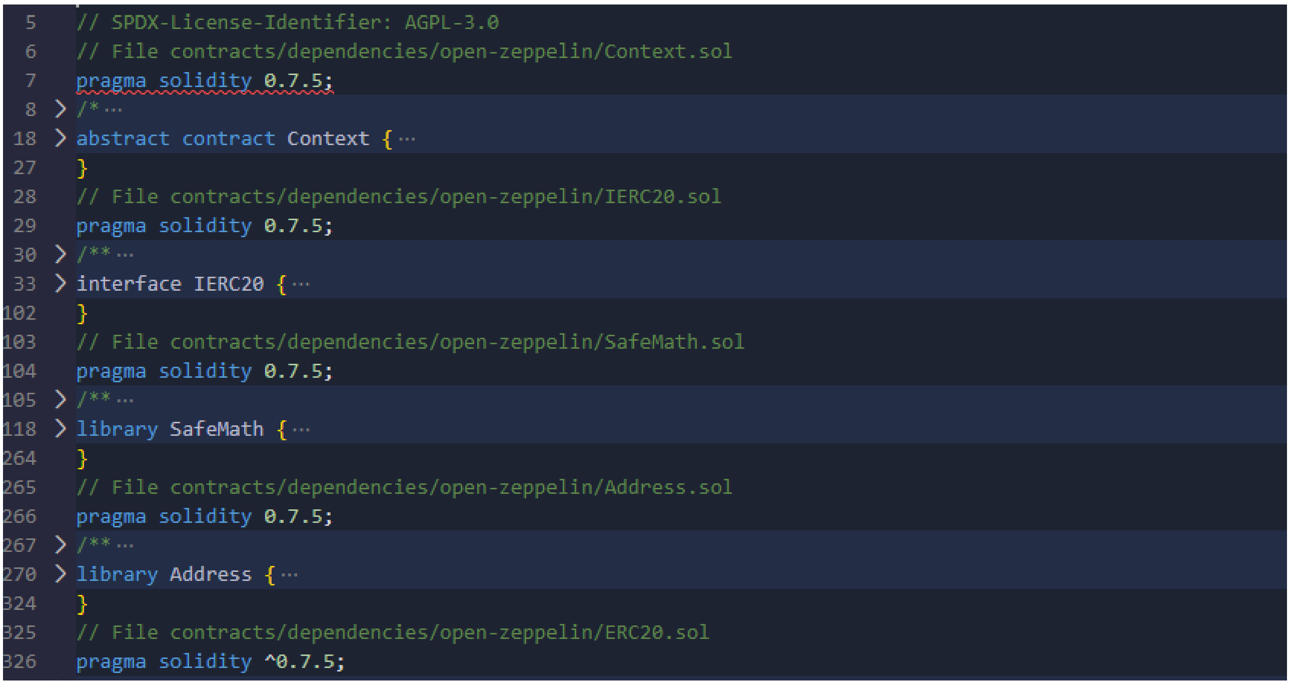
Very Low / Informational / Best practices:

(1) block timestamp:

Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.

Resolution: Avoid relying on block.timestamp.

(2) Multiple pragma:

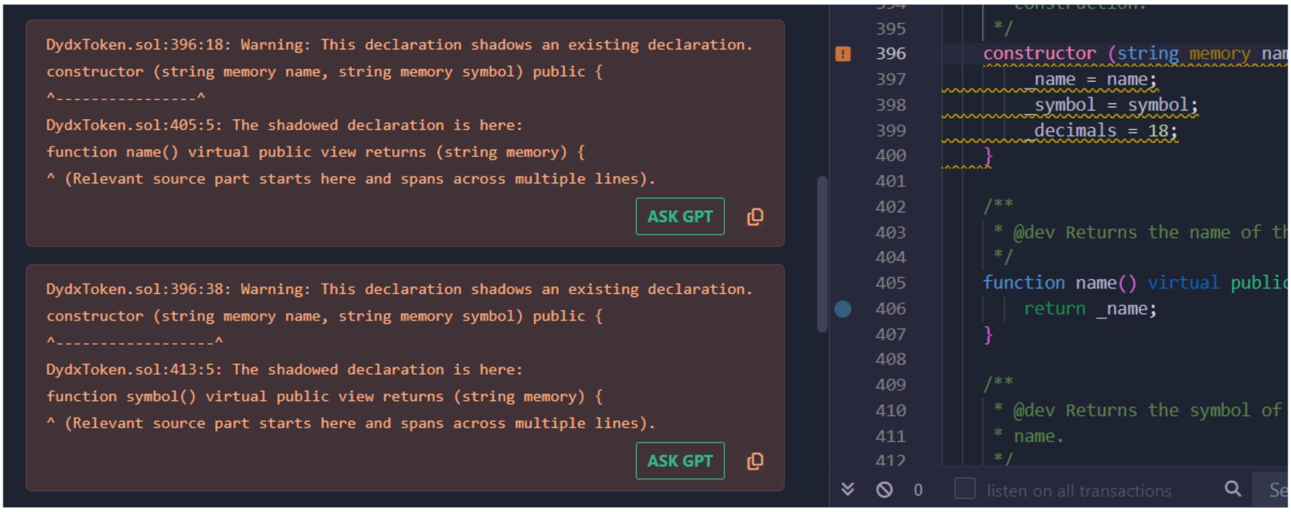


```
// SPDX-License-Identifier: AGPL-3.0
// File contracts/dependencies/open-zeppelin/Context.sol
pragma solidity 0.7.5;
/*…
abstract contract Context {…
}
// File contracts/dependencies/open-zeppelin/IERC20.sol
pragma solidity 0.7.5;
/**…
interface IERC20 {…
}
// File contracts/dependencies/open-zeppelin/SafeMath.sol
pragma solidity 0.7.5;
/**…
library SafeMath {…
}
// File contracts/dependencies/open-zeppelin/Address.sol
pragma solidity 0.7.5;
/**…
library Address {…
}
// File contracts/dependencies/open-zeppelin/ERC20.sol
pragma solidity ^0.7.5;
```

Multiple pragmas are deducted.

Resolution: We suggest using one solidity pragma.

(3) Compile time warning:



```
DydxToken.sol:396:18: Warning: This declaration shadows an existing declaration.
constructor (string memory name, string memory symbol) public {
^---------------^
DydxToken.sol:405:5: The shadowed declaration is here:
function name() virtual public view returns (string memory) {
^ (Relevant source part starts here and spans across multiple lines).
```

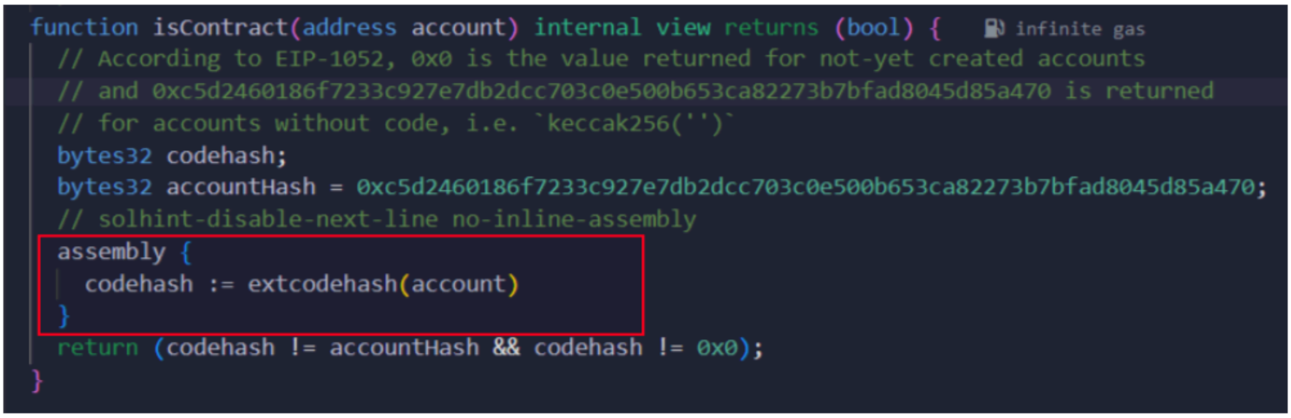
```
DydxToken.sol:396:38: Warning: This declaration shadows an existing declaration.
constructor (string memory name, string memory symbol) public {
^-----------------^
DydxToken.sol:413:5: The shadowed declaration is here:
function symbol() virtual public view returns (string memory) {
^ (Relevant source part starts here and spans across multiple lines).
```

```
 */
constructor (string memory nam
    _name = name;
    _symbol = symbol;
    _decimals = 18;
}

/**
 * @dev Returns the name of th
 */
function name() virtual public
    return _name;
}

/**
 * @dev Returns the symbol of
 * name.
 */
```

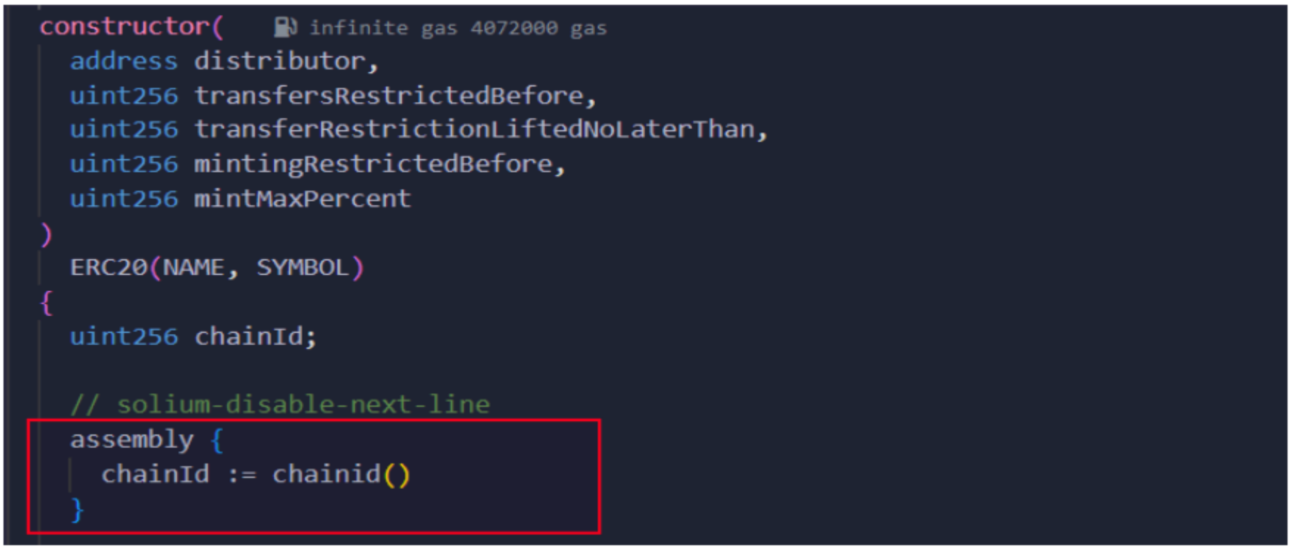
Constructor argument, name, and symbol is a shadow declaration of name and symbol, function name.

Resolution: We suggest changing the constructor argument variable names like _tokenName and _tokenSymbol.

(4) Solidity assembly:



```
function isContract(address account) internal view returns (bool) {   infinite gas
  // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
  // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
  // for accounts without code, i.e. `keccak256('')`
  bytes32 codehash;
  bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
  // solhint-disable-next-line no-inline-assembly
  assembly {
    codehash := extcodehash(account)
  }
  return (codehash != accountHash && codehash != 0x0);
}
```

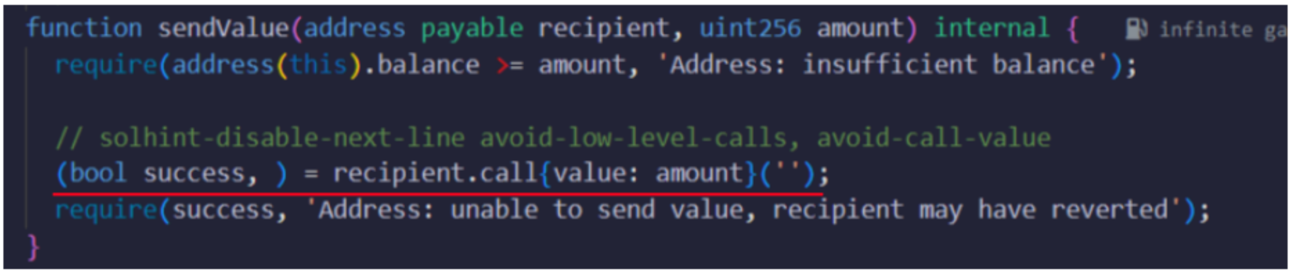


```
constructor(   infinite gas 4072000 gas
  address distributor,
  uint256 transfersRestrictedBefore,
  uint256 transferRestrictionLiftedNoLaterThan,
  uint256 mintingRestrictedBefore,
  uint256 mintMaxPercent
)
  ERC20(NAME, SYMBOL)
{
  uint256 chainId;

  // solium-disable-next-line
  assembly {
    chainId := chainid()
  }
```

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Resolution: It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

(5) Use of "call": should be avoided whenever possible:

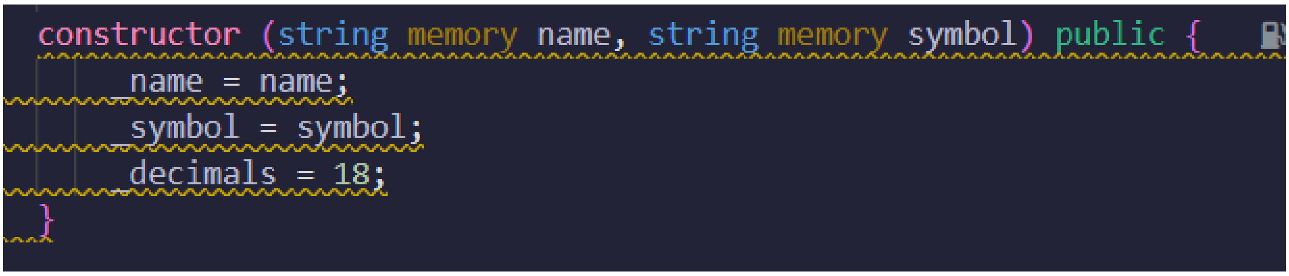


```
function sendValue(address payable recipient, uint256 amount) internal {   infinite ga
  require(address(this).balance >= amount, 'Address: insufficient balance');

  // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
  (bool success, ) = recipient.call{value: amount}('');
  require(success, 'Address: unable to send value, recipient may have reverted');
}
```

It can lead to unexpected behavior if the return value is not handled properly.

Resolution: Please use Direct Calls by specifying the called contract's interface.

(6) Coding style:



```
constructor (string memory name, string memory symbol) public {
        _name = name;
        _symbol = symbol;
        _decimals = 18;
}
```

Explicitly specifying the public for a constructor in this version generates a warning, as the visibility is ignored.

Resolution: We suggest removing the Explicit specifier 'public' visibility keyword.

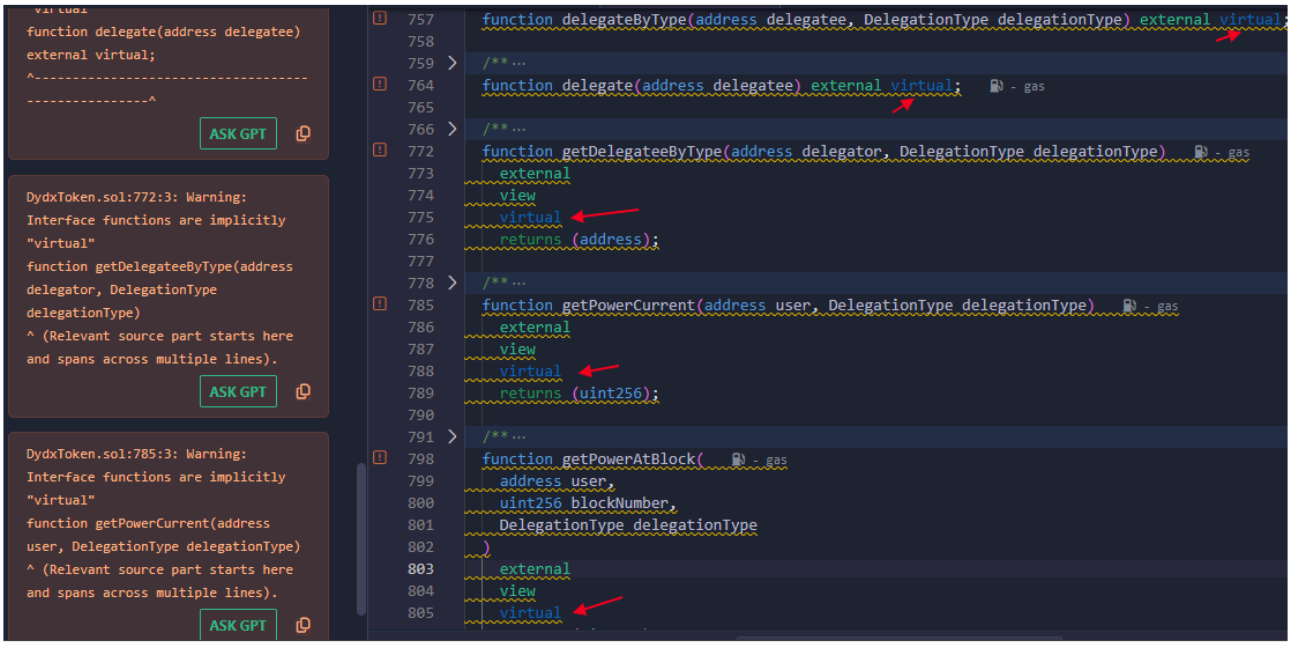
(7) Centralization:

In the contract onlyOwner() is an owner authority on the following function:

- renounceOwnership
- transferOwnership
- addToTokenTransferAllowlist
- removeFromTokenTransferAllowlist
- updateTransfersRestrictedBefore
- mint

Resolution: We suggest carefully managing the onlyOwner private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices.

(8) Redundant 'virtual' keyword:



```
"virtual"
function delegate(address delegatee)
external virtual;
^-----------------------------------
---------------^
```

```
DydxToken.sol:772:3: Warning:
Interface functions are implicitly
"virtual"
function getDelegateeByType(address
delegator, DelegationType
delegationType)
^ (Relevant source part starts here
and spans across multiple lines).
```

```
DydxToken.sol:785:3: Warning:
Interface functions are implicitly
"virtual"
function getPowerCurrent(address
user, DelegationType delegationType)
^ (Relevant source part starts here
and spans across multiple lines).
```

```
function delegateByType(address delegatee, DelegationType delegationType) external virtual;

/**...
function delegate(address delegatee) external virtual;    - gas

/**...
function getDelegateeByType(address delegator, DelegationType delegationType)    - gas
  external
  view
  virtual
  returns (address);

/**...
function getPowerCurrent(address user, DelegationType delegationType)    - gas
  external
  view
  virtual
  returns (uint256);

/**...
function getPowerAtBlock(    - gas
  address user,
  uint256 blockNumber,
  DelegationType delegationType
)
  external
  view
  virtual
```

In Solidity version 0.7.5 and later, all functions in an interface are considered virtual by default, and explicitly using the virtual keyword is not necessary.

Resolution: We suggest simply removing the 'virtual' keyword from the function declarations in your interface.

(9) Visibility can be external over the public:

Any functions which are not called internally should be declared as external. This saves some gas and is considered a good practice.

https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

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. The following are Admin functions:

DydxToken.sol

- addToTokenTransferAllowlist: Adds addresses to the token transfer allowlist only callable by the owner.
- removeFromTokenTransferAllowlist: Removes addresses from the token transfer allowlist only callable by the owner.
- updateTransfersRestrictedBefore: Updates the transfer restriction only callable by the owner.
- mint: Mint new tokens are only callable by the owner after the required time period has elapsed.
- permit: Implements the permit function only callable by the owner.

Ownable.sol

- renounceOwnership: Deleting ownership will leave the contract without an owner, removing any owner-only functionality.
- transferOwnership: Current owner can transfer ownership of the contract to a new account.

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 Etherscan web links. And we have used all possible tests based on given objects as files. We observed 1 medium And 9 Informational issues 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, 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).

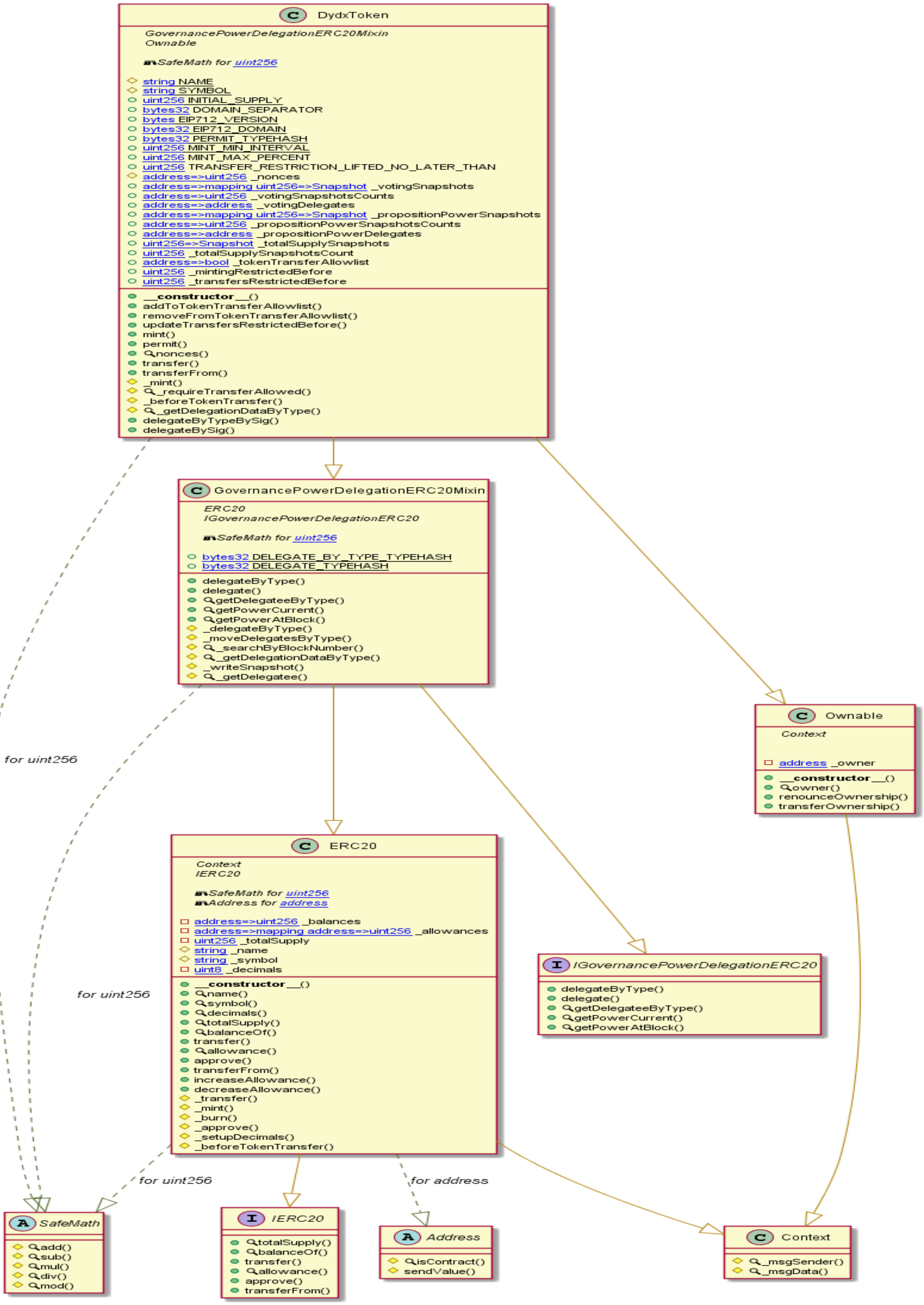
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While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

Appendix

Code Flow Diagram - dYdX Token

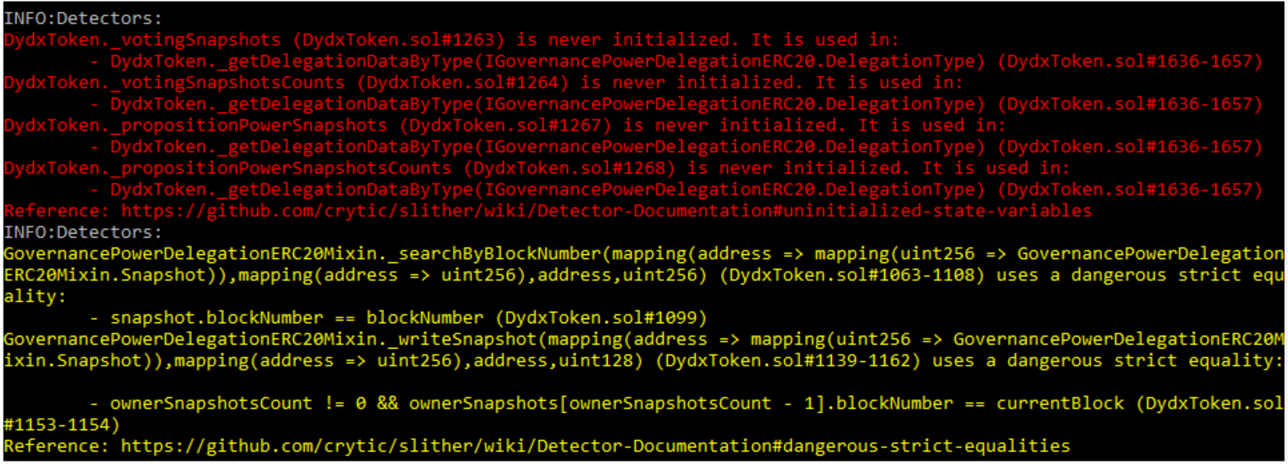


Slither Results Log

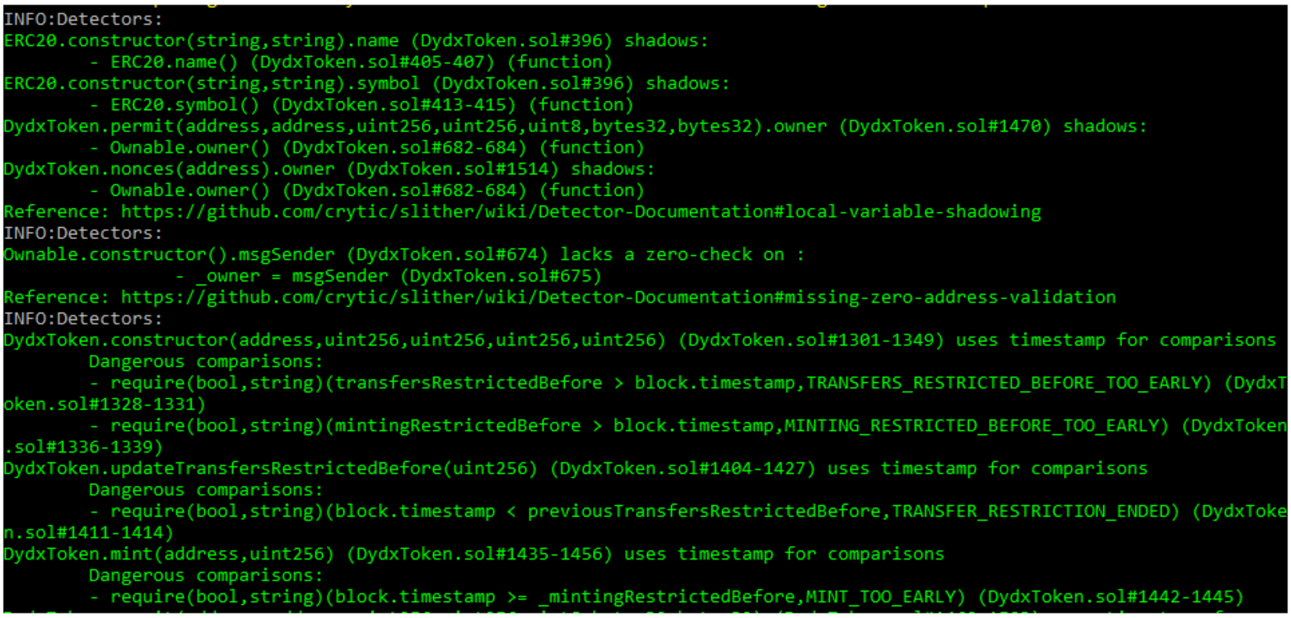
Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report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

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

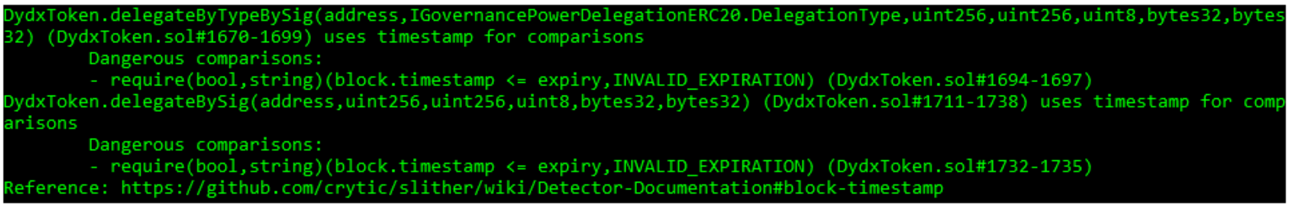
Slither Log >> DydxToken.sol



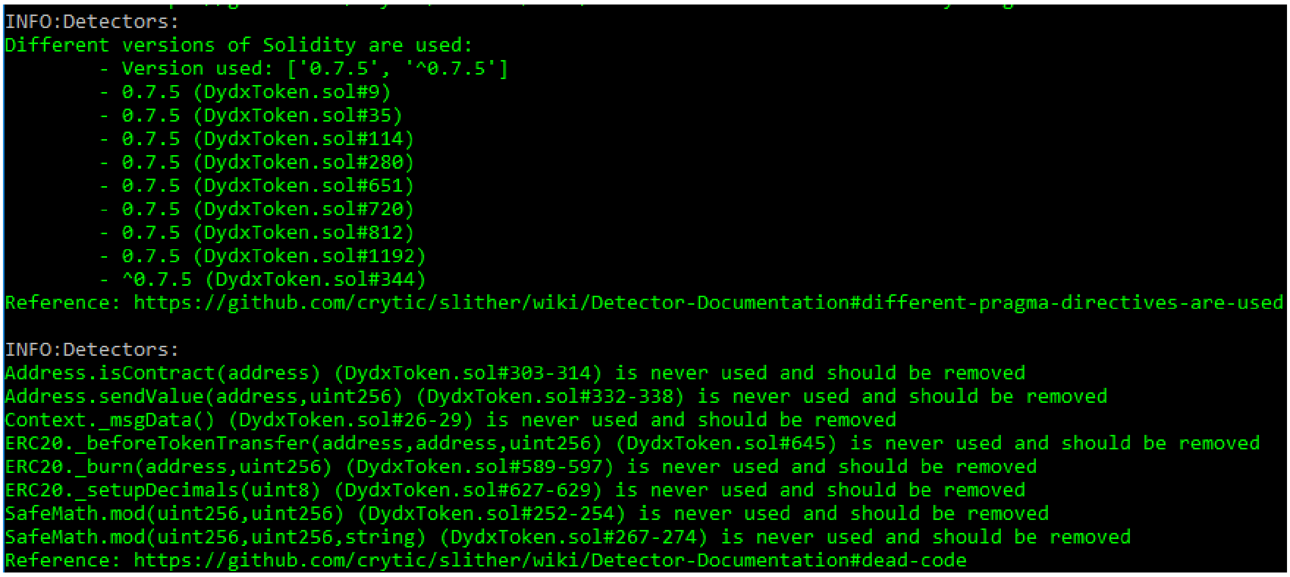
```
INFO:Detectors:
DydxToken._votingSnapshots (DydxToken.sol#1263) is never initialized. It is used in:
	- DydxToken._getDelegationDataByType(IGovernancePowerDelegationERC20.DelegationType) (DydxToken.sol#1636-1657)
DydxToken._votingSnapshotsCounts (DydxToken.sol#1264) is never initialized. It is used in:
	- DydxToken._getDelegationDataByType(IGovernancePowerDelegationERC20.DelegationType) (DydxToken.sol#1636-1657)
DydxToken._propositionPowerSnapshots (DydxToken.sol#1267) is never initialized. It is used in:
	- DydxToken._getDelegationDataByType(IGovernancePowerDelegationERC20.DelegationType) (DydxToken.sol#1636-1657)
DydxToken._propositionPowerSnapshotsCounts (DydxToken.sol#1268) is never initialized. It is used in:
	- DydxToken._getDelegationDataByType(IGovernancePowerDelegationERC20.DelegationType) (DydxToken.sol#1636-1657)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
INFO:Detectors:
GovernancePowerDelegationERC20Mixin._searchByBlockNumber(mapping(address => mapping(uint256 => GovernancePowerDelegation
ERC20Mixin.Snapshot)),mapping(address => uint256),address,uint256) (DydxToken.sol#1063-1108) uses a dangerous strict equ
ality:
	- snapshot.blockNumber == blockNumber (DydxToken.sol#1099)
GovernancePowerDelegationERC20Mixin._writeSnapshot(mapping(address => mapping(uint256 => GovernancePowerDelegationERC20M
ixin.Snapshot)),mapping(address => uint256),address,uint128) (DydxToken.sol#1139-1162) uses a dangerous strict equality:

	- ownerSnapshotsCount != 0 && ownerSnapshots[ownerSnapshotsCount - 1].blockNumber == currentBlock (DydxToken.sol
#1153-1154)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
```



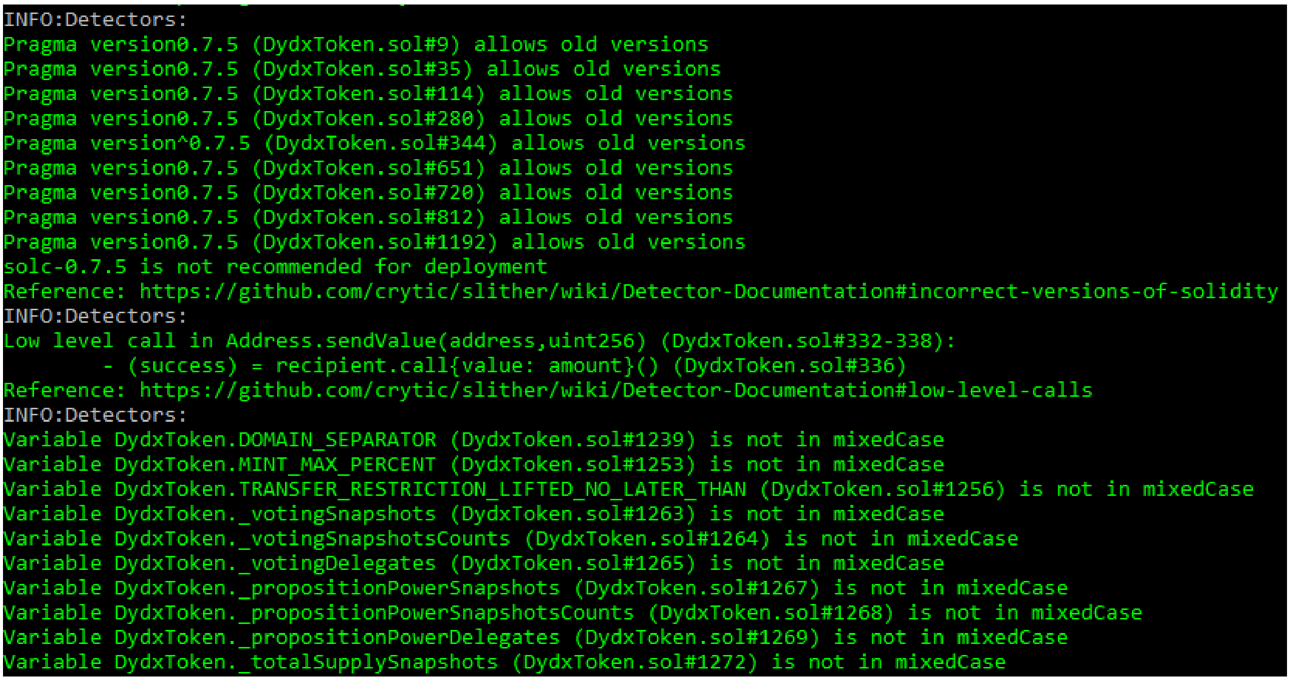
```
INFO:Detectors:
ERC20.constructor(string,string).name (DydxToken.sol#396) shadows:
	- ERC20.name() (DydxToken.sol#405-407) (function)
ERC20.constructor(string,string).symbol (DydxToken.sol#396) shadows:
	- ERC20.symbol() (DydxToken.sol#413-415) (function)
DydxToken.permit(address,address,uint256,uint256,uint8,bytes32,bytes32).owner (DydxToken.sol#1470) shadows:
	- Ownable.owner() (DydxToken.sol#682-684) (function)
DydxToken.nonces(address).owner (DydxToken.sol#1514) shadows:
	- Ownable.owner() (DydxToken.sol#682-684) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Ownable.constructor().msgSender (DydxToken.sol#674) lacks a zero-check on :
		- _owner = msgSender (DydxToken.sol#675)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
DydxToken.constructor(address,uint256,uint256,uint256,uint256) (DydxToken.sol#1301-1349) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(transfersRestrictedBefore > block.timestamp,TRANSFERS_RESTRICTED_BEFORE_TOO_EARLY) (DydxT
oken.sol#1328-1331)
	- require(bool,string)(mintingRestrictedBefore > block.timestamp,MINTING_RESTRICTED_BEFORE_TOO_EARLY) (DydxToken
.sol#1336-1339)
DydxToken.updateTransfersRestrictedBefore(uint256) (DydxToken.sol#1404-1427) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(block.timestamp < previousTransfersRestrictedBefore,TRANSFER_RESTRICTION_ENDED) (DydxToke
n.sol#1411-1414)
DydxToken.mint(address,uint256) (DydxToken.sol#1435-1456) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(block.timestamp >= _mintingRestrictedBefore,MINT_TOO_EARLY) (DydxToken.sol#1442-1445)
```



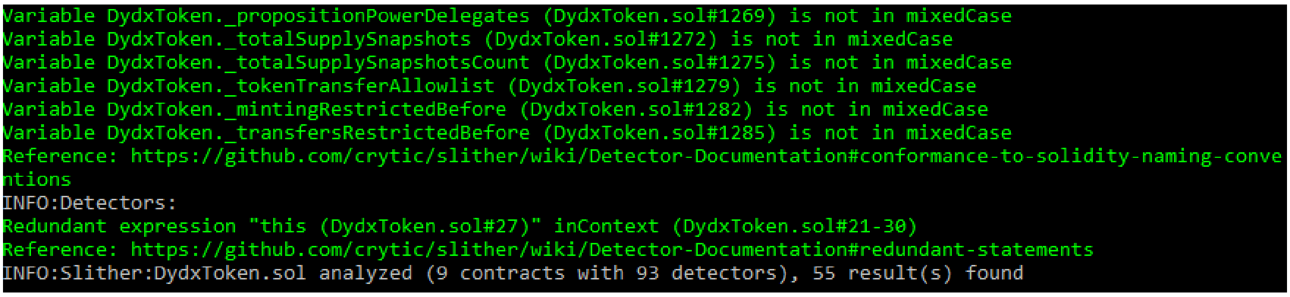
```
DydxToken.delegateByTypeBySig(address,IGovernancePowerDelegationERC20.DelegationType,uint256,uint256,uint8,bytes32,bytes
32) (DydxToken.sol#1670-1699) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(block.timestamp <= expiry,INVALID_EXPIRATION) (DydxToken.sol#1694-1697)
DydxToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (DydxToken.sol#1711-1738) uses timestamp for comp
arisons
	Dangerous comparisons:
	- require(bool,string)(block.timestamp <= expiry,INVALID_EXPIRATION) (DydxToken.sol#1732-1735)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```



```
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['0.7.5', '^0.7.5']
        - 0.7.5 (DydxToken.sol#9)
        - 0.7.5 (DydxToken.sol#35)
        - 0.7.5 (DydxToken.sol#114)
        - 0.7.5 (DydxToken.sol#280)
        - 0.7.5 (DydxToken.sol#651)
        - 0.7.5 (DydxToken.sol#720)
        - 0.7.5 (DydxToken.sol#812)
        - 0.7.5 (DydxToken.sol#1192)
        - ^0.7.5 (DydxToken.sol#344)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

INFO:Detectors:
Address.isContract(address) (DydxToken.sol#303-314) is never used and should be removed
Address.sendValue(address,uint256) (DydxToken.sol#332-338) is never used and should be removed
Context._msgData() (DydxToken.sol#26-29) is never used and should be removed
ERC20._beforeTokenTransfer(address,address,uint256) (DydxToken.sol#645) is never used and should be removed
ERC20._burn(address,uint256) (DydxToken.sol#589-597) is never used and should be removed
ERC20._setupDecimals(uint8) (DydxToken.sol#627-629) is never used and should be removed
SafeMath.mod(uint256,uint256) (DydxToken.sol#252-254) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (DydxToken.sol#267-274) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```



```
INFO:Detectors:
Pragma version0.7.5 (DydxToken.sol#9) allows old versions
Pragma version0.7.5 (DydxToken.sol#35) allows old versions
Pragma version0.7.5 (DydxToken.sol#114) allows old versions
Pragma version0.7.5 (DydxToken.sol#280) allows old versions
Pragma version^0.7.5 (DydxToken.sol#344) allows old versions
Pragma version0.7.5 (DydxToken.sol#651) allows old versions
Pragma version0.7.5 (DydxToken.sol#720) allows old versions
Pragma version0.7.5 (DydxToken.sol#812) allows old versions
Pragma version0.7.5 (DydxToken.sol#1192) allows old versions
solc-0.7.5 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (DydxToken.sol#332-338):
        - (success) = recipient.call{value: amount}() (DydxToken.sol#336)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Variable DydxToken.DOMAIN_SEPARATOR (DydxToken.sol#1239) is not in mixedCase
Variable DydxToken.MINT_MAX_PERCENT (DydxToken.sol#1253) is not in mixedCase
Variable DydxToken.TRANSFER_RESTRICTION_LIFTED_NO_LATER_THAN (DydxToken.sol#1256) is not in mixedCase
Variable DydxToken._votingSnapshots (DydxToken.sol#1263) is not in mixedCase
Variable DydxToken._votingSnapshotsCounts (DydxToken.sol#1264) is not in mixedCase
Variable DydxToken._votingDelegates (DydxToken.sol#1265) is not in mixedCase
Variable DydxToken._propositionPowerSnapshots (DydxToken.sol#1267) is not in mixedCase
Variable DydxToken._propositionPowerSnapshotsCounts (DydxToken.sol#1268) is not in mixedCase
Variable DydxToken._propositionPowerDelegates (DydxToken.sol#1269) is not in mixedCase
Variable DydxToken._totalSupplySnapshots (DydxToken.sol#1272) is not in mixedCase
```

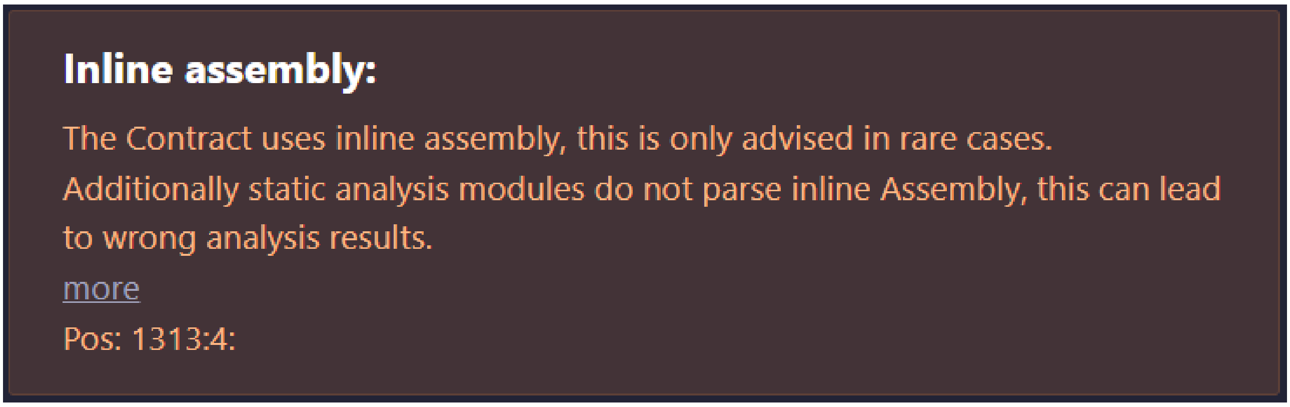


```
Variable DydxToken._propositionPowerDelegates (DydxToken.sol#1269) is not in mixedCase
Variable DydxToken._totalSupplySnapshots (DydxToken.sol#1272) is not in mixedCase
Variable DydxToken._totalSupplySnapshotsCount (DydxToken.sol#1275) is not in mixedCase
Variable DydxToken._tokenTransferAllowlist (DydxToken.sol#1279) is not in mixedCase
Variable DydxToken._mintingRestrictedBefore (DydxToken.sol#1282) is not in mixedCase
Variable DydxToken._transfersRestrictedBefore (DydxToken.sol#1285) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conve
ntions
INFO:Detectors:
Redundant expression "this (DydxToken.sol#27)" inContext (DydxToken.sol#21-30)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Slither:DydxToken.sol analyzed (9 contracts with 93 detectors), 55 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.

DydxToken.sol

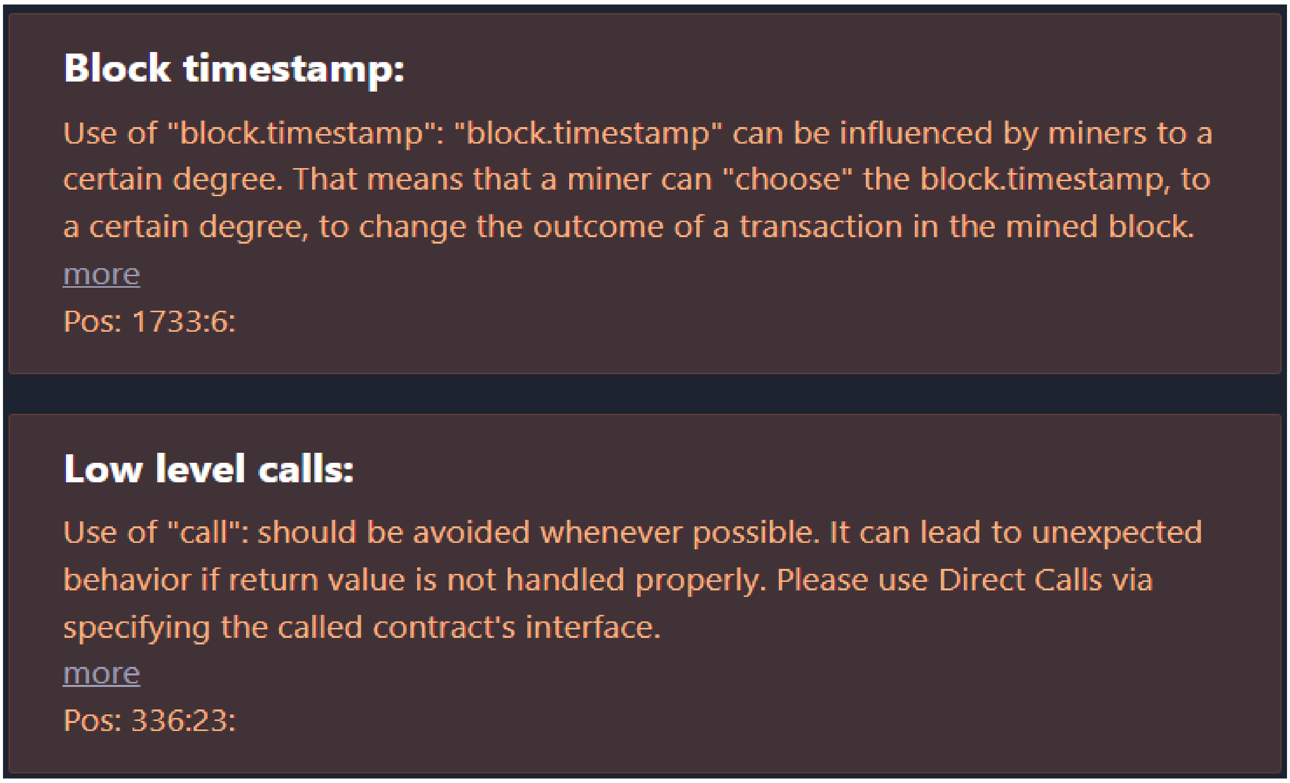


Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

more

Pos: 1313:4:



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.

more

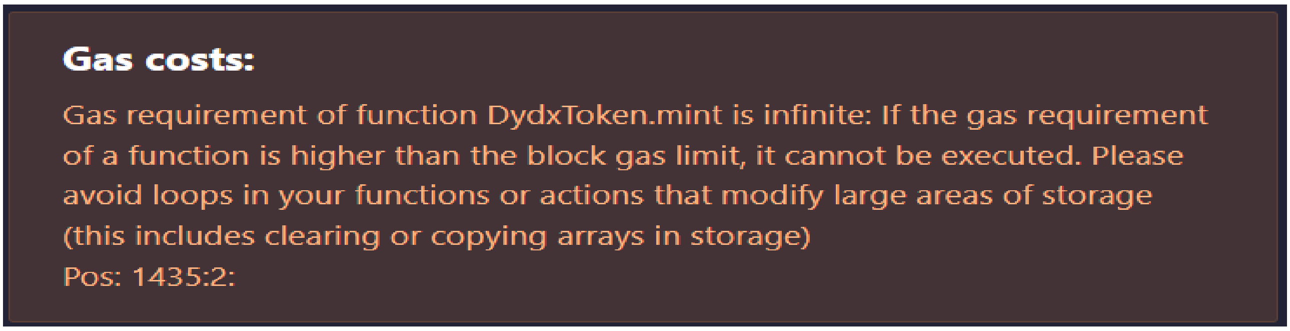
Pos: 1733:6:

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

more

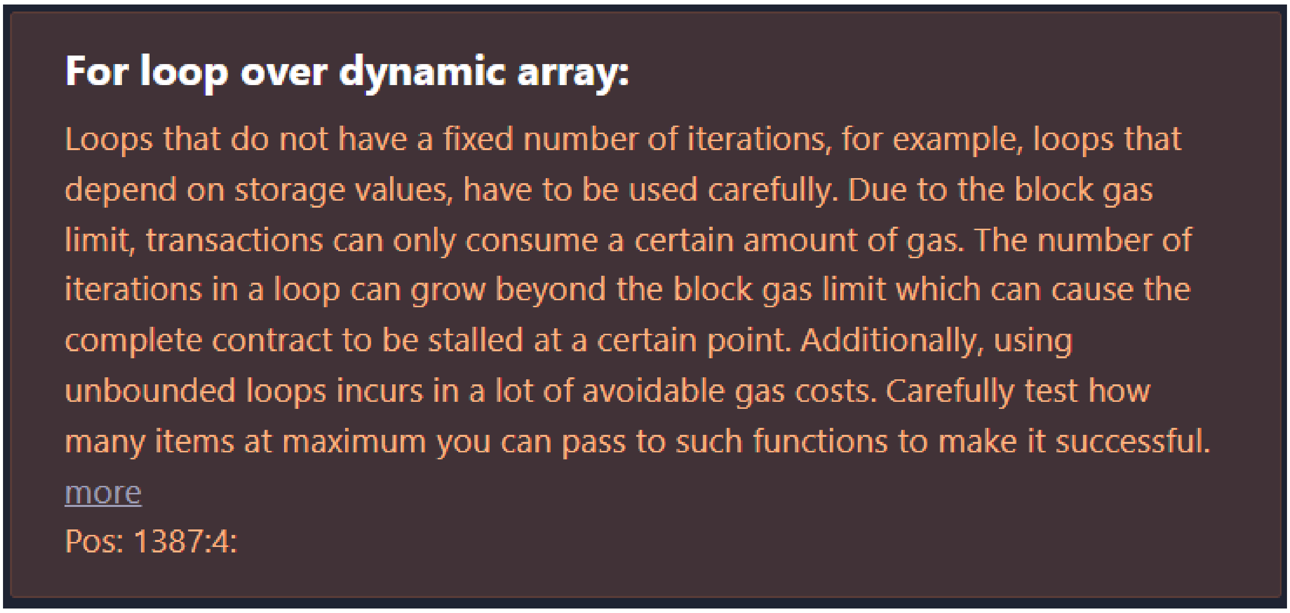
Pos: 336:23:



Gas costs:

Gas requirement of function DydxToken.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: 1435:2:

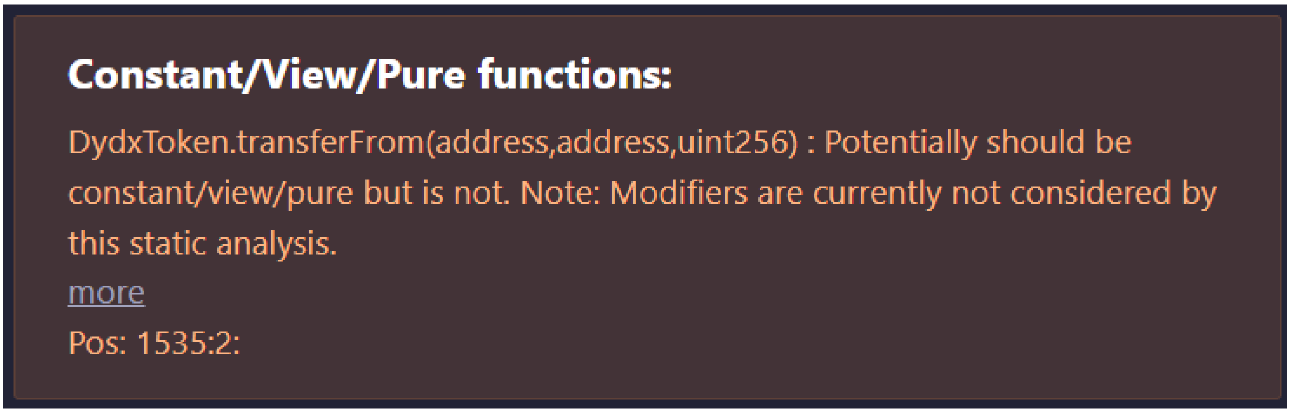


For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

more

Pos: 1387:4:

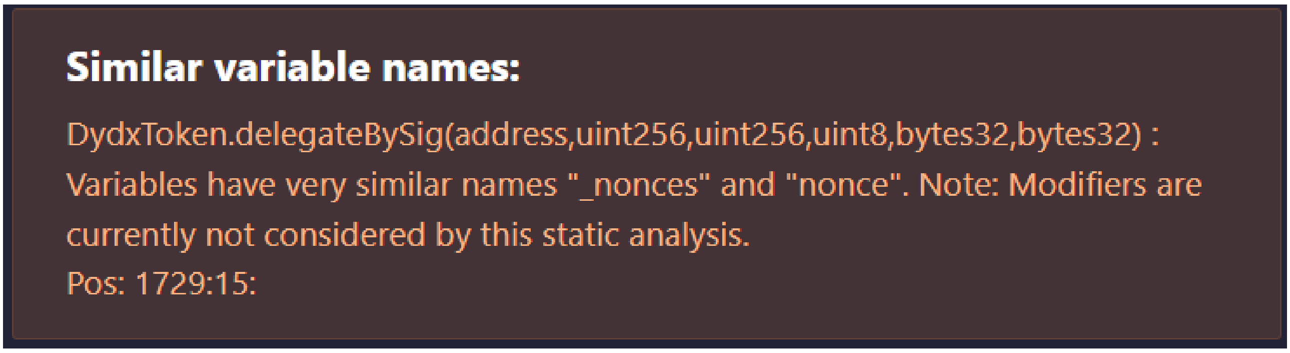


Constant/View/Pure functions:

DydxToken.transferFrom(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

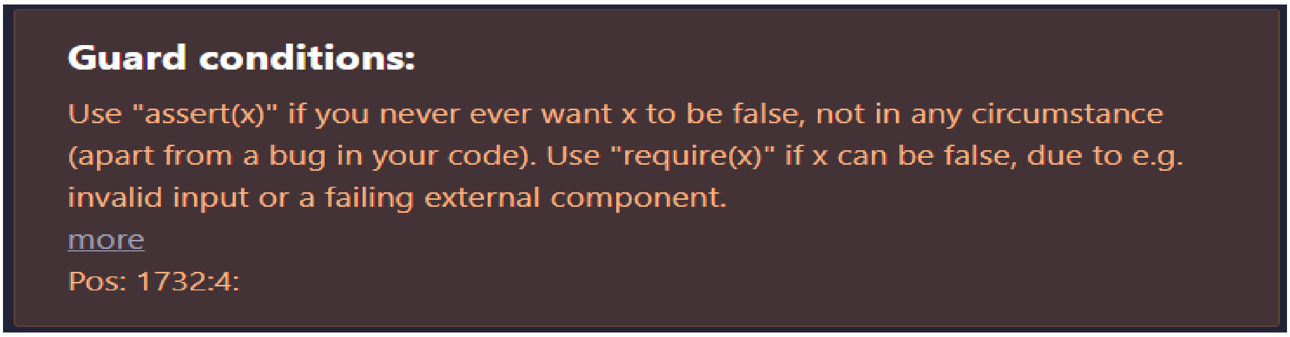
Pos: 1535:2:



Similar variable names:

DydxToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) : Variables have very similar names "_nonces" and "nonce". Note: Modifiers are currently not considered by this static analysis.

Pos: 1729:15:

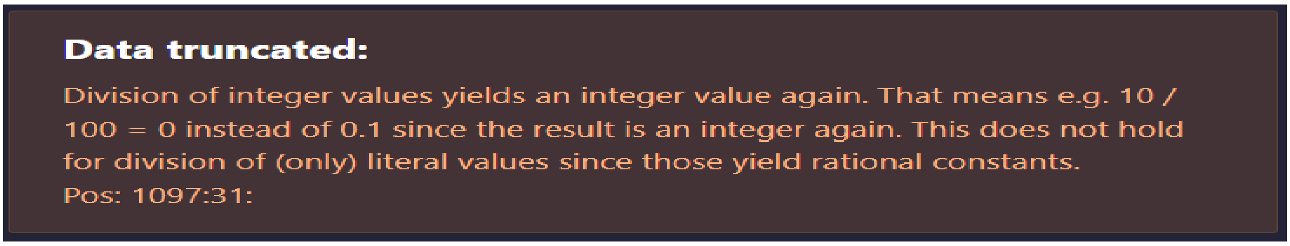


Guard conditions:

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.

more

Pos: 1732:4:



Data truncated:

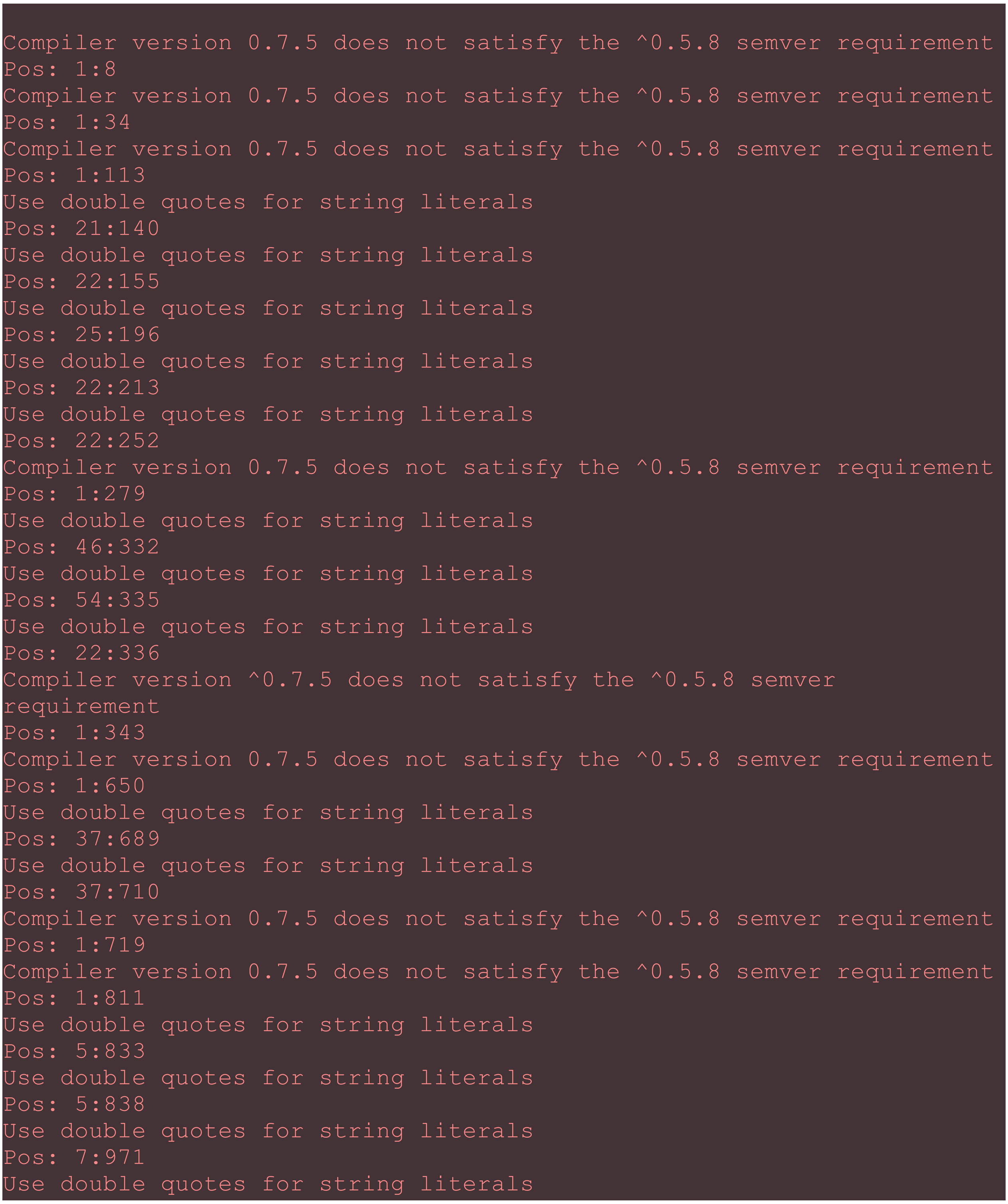
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 1097:31:

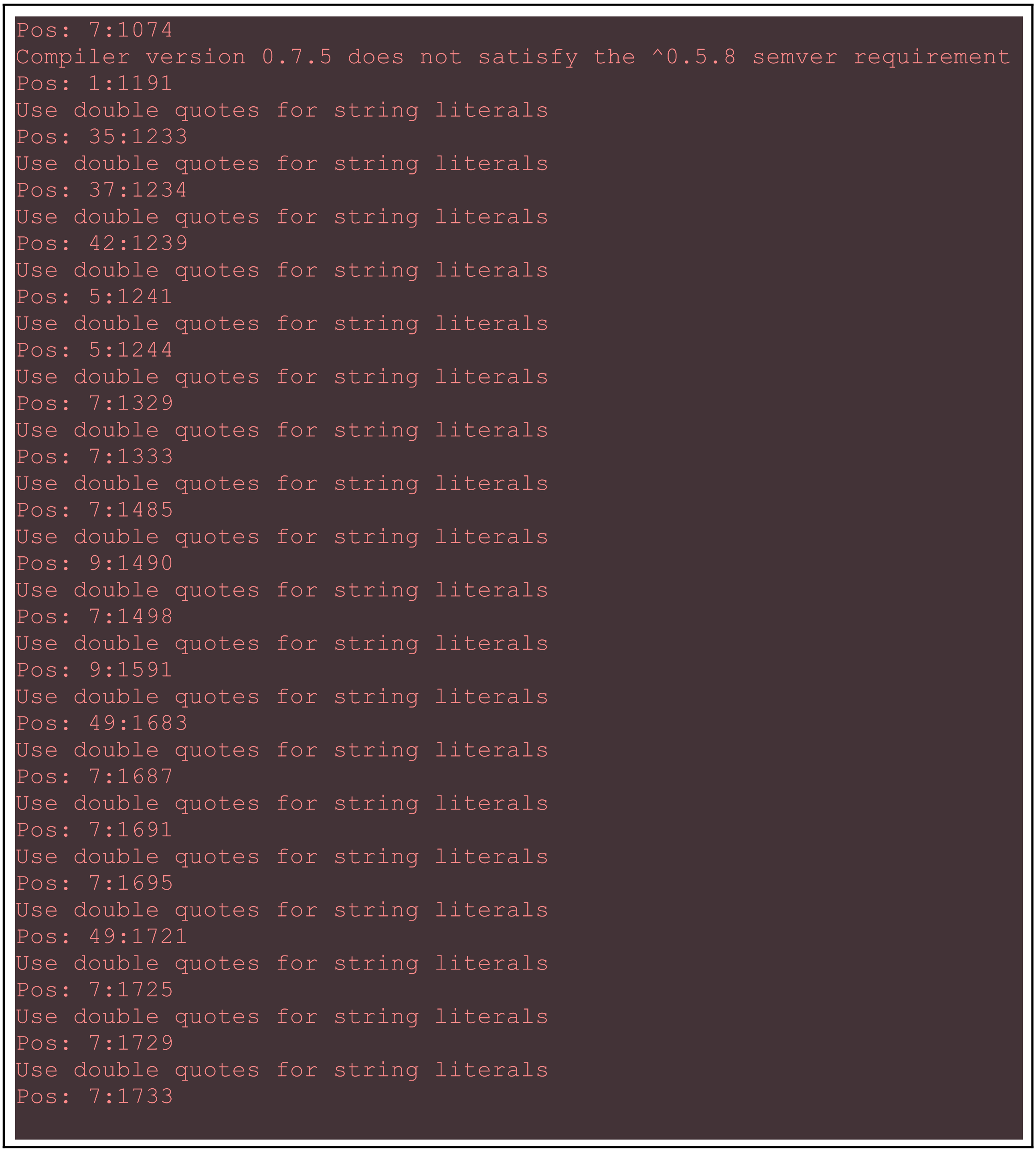
Solhint Linter

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.

DydxToken.sol



```
Compiler version 0.7.5 does not satisfy the ^0.5.8 semver requirement
Pos: 1:8
Compiler version 0.7.5 does not satisfy the ^0.5.8 semver requirement
Pos: 1:34
Compiler version 0.7.5 does not satisfy the ^0.5.8 semver requirement
Pos: 1:113
Use double quotes for string literals
Pos: 21:140
Use double quotes for string literals
Pos: 22:155
Use double quotes for string literals
Pos: 25:196
Use double quotes for string literals
Pos: 22:213
Use double quotes for string literals
Pos: 22:252
Compiler version 0.7.5 does not satisfy the ^0.5.8 semver requirement
Pos: 1:279
Use double quotes for string literals
Pos: 46:332
Use double quotes for string literals
Pos: 54:335
Use double quotes for string literals
Pos: 22:336
Compiler version ^0.7.5 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:343
Compiler version 0.7.5 does not satisfy the ^0.5.8 semver requirement
Pos: 1:650
Use double quotes for string literals
Pos: 37:689
Use double quotes for string literals
Pos: 37:710
Compiler version 0.7.5 does not satisfy the ^0.5.8 semver requirement
Pos: 1:719
Compiler version 0.7.5 does not satisfy the ^0.5.8 semver requirement
Pos: 1:811
Use double quotes for string literals
Pos: 5:833
Use double quotes for string literals
Pos: 5:838
Use double quotes for string literals
Pos: 7:971
Use double quotes for string literals
```



```
Pos: 7:1074
Compiler version 0.7.5 does not satisfy the ^0.5.8 semver requirement
Pos: 1:1191
Use double quotes for string literals
Pos: 35:1233
Use double quotes for string literals
Pos: 37:1234
Use double quotes for string literals
Pos: 42:1239
Use double quotes for string literals
Pos: 5:1241
Use double quotes for string literals
Pos: 5:1244
Use double quotes for string literals
Pos: 7:1329
Use double quotes for string literals
Pos: 7:1333
Use double quotes for string literals
Pos: 7:1485
Use double quotes for string literals
Pos: 9:1490
Use double quotes for string literals
Pos: 7:1498
Use double quotes for string literals
Pos: 9:1591
Use double quotes for string literals
Pos: 49:1683
Use double quotes for string literals
Pos: 7:1687
Use double quotes for string literals
Pos: 7:1691
Use double quotes for string literals
Pos: 7:1695
Use double quotes for string literals
Pos: 49:1721
Use double quotes for string literals
Pos: 7:1725
Use double quotes for string literals
Pos: 7:1729
Use double quotes for string literals
Pos: 7:1733
```

Software analysis result:

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