EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Plutus Land
Website:	http://plutuslandofficial.com/
Platform:	Ethereum
Language:	Solidity
Date:	July 29th, 2022

Table of contents

`

Introduction

EtherAuthority was contracted by the Plutus Land team to perform the Security audit of the Plutus Land smart contract code. 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 July 29th, 2022.

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

- Plutus Land is a NFT smart contract, having functions like mint, setBaseURI, refund, nftAdd, nftSub, etc.
- The Key4 Token contract inherits ERC721, ERC721Enumerable, Address, ERC721, ERC721EnuCounters, Pausable, Ownable standard smart contracts from the OpenZeppelin library.
- These OpenZeppelin contracts are considered community- audited and time-tested, and hence are not part of the audit scope.

Audit scope

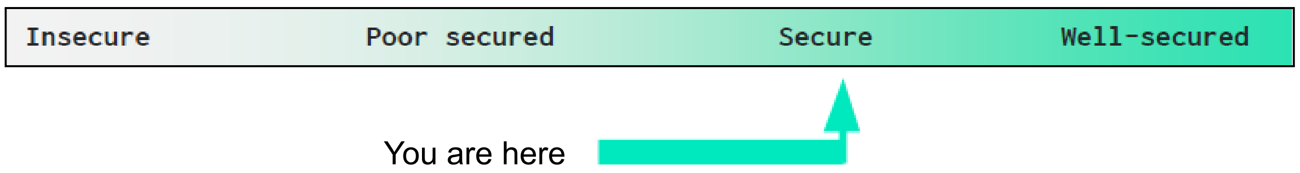
Name	**Code Review and Security Analysis Report for Plutus Land Smart Contract**
Platform	**Etherscan / Solidity**
File	Mint.sol
File MD5 Hash	3551DE6782654F2207B2650187A93BE4
Audit Date	July 29th, 2022
Revised Audit Date	August 1st, 2022

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Name: Key4 ● Symbol: KEY4 ● Price Per Token: 0.8 ether	YES, This is valid.
Ownership Control: ● Owner can set the recipient of revenues. ● Owner allows to change the mint price.	YES, This is valid.
Tokens are non-transferable.	YES, This is valid.

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **"Secured".** This token contract does contain owner control, which does not make it fully decentralized.



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on 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. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 1 high, 0 medium and 1 low and some very low level issues.
All the issues have been resolved/acknowledged in the revised code.

Investors Advice: 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	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	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**

Code Quality

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

The libraries in Key4 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 Key4 Token.

The Plutus Land 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 Key4 Token smart contract code in the form of a file. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **well** commented. 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 used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	whenNotPaused	modifier	Unused local libraries	Refer audit findings
3	whenPaused	modifier	Passed	No Issue
4	paused	read	Passed	No Issue
5	_requireNotPaused	internal	Passed	No Issue
6	_requirePaused	internal	Passed	No Issue
7	_pause	internal	Passed	No Issue
8	_unpause	internal	Passed	No Issue
9	owner	read	Passed	No Issue
10	onlyOwner	modifier	Passed	No Issue
11	renounceOwnership	write	access only Owner	No Issue
12	transferOwnership	write	access only Owner	No Issue
13	_setOwner	write	Passed	No Issue
14	supportsInterface	read	Passed	No Issue
15	balanceOf	read	Passed	No Issue
16	ownerOf	read	Passed	No Issue
17	name	read	Passed	No Issue
18	symbol	read	Passed	No Issue
19	tokenURI	read	Passed	No Issue
20	_baseURI	internal	Passed	No Issue
21	approve	write	Passed	No Issue
22	getApproved	read	Passed	No Issue
23	setApprovalForAll	write	Passed	No Issue
24	isApprovedForAll	read	Passed	No Issue
25	transferFrom	write	Passed	No Issue
26	safeTransferFrom	write	Passed	No Issue
27	safeTransferFrom	write	Passed	No Issue
28	_safeTransfer	internal	Passed	No Issue
29	_exists	internal	Passed	No Issue
30	_isApprovedOrOwner	internal	Passed	No Issue
31	_safeMint	internal	Passed	No Issue
32	_safeMint	internal	Passed	No Issue
33	_mint	internal	Passed	No Issue
34	_transfer	internal	Passed	No Issue
35	_burn	internal	Passed	No Issue
36	_approve	internal	Passed	No Issue
37	_setApprovalForAll	internal	Passed	No Issue
38	_requireMinted	internal	Passed	No Issue
39	_checkOnERC721Received	write	Passed	No Issue
40	_beforeTokenTransfer	internal	Passed	No Issue
41	_afterTokenTransfer	internal	Passed	No Issue

42	mint	external	Passed	No Issue
43	_multimint	write	Passed	No Issue
44	setBeneficiary	write	Function input parameters lack of check	Refer audit findings
45	_checkOwner	internal	Passed	No Issue
46	getPrice	read	Passed	No Issue
47	setPrice	write	Function input parameters lack of check	Refer audit findings
48	totalSupply	read	Passed	No Issue
49	tokenURI	read	Passed	No Issue
50	getBaseURI	read	Passed	No Issue
51	setBaseURI	write	Passed	No Issue
52	_baseURI	internal	Passed	No Issue
53	supportsInterface	read	Passed	No Issue
54	_beforeTokenTransfer	internal	Passed	No Issue
55	refund	write	Passed	No Issue
56	gettokenID	read	Passed	No Issue
57	getnftamount	read	Passed	No Issue
58	nftAdd	internal	Passed	No Issue
59	setApprovalForAll	write	Passed	No Issue
60	nftSub	internal	Passed	No Issue
61	transferFrom	write	Passed	No Issue
62	safeTransferFrom	write	Passed	No Issue
63	safeTransferFrom	write	Passed	No Issue
64	approve	write	Passed	No Issue
65	transferOwnership	write	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 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 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.

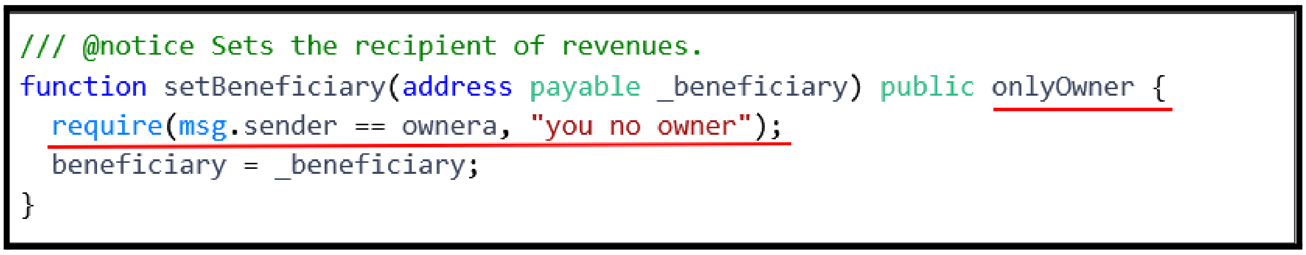
Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

(1) The onlyOwner Modifier not working:



```
/// @notice Sets the recipient of revenues.
function setBeneficiary(address payable _beneficiary) public onlyOwner {
  require(msg.sender == ownera, "you no owner");
  beneficiary = _beneficiary;
}
```

Ownable library has been included and code using onlyOwner modifier, but it's not working as the _checkOwner function has been overridden as blank. So anyone can execute the owner's function.

The onlyOwner functions are:

- setBeneficiary()
- setPrice()
- setBaseURI()

Resolution: We suggest removing that _checkOwner override function from the contract, so the ownerOnly modifier will work.

Status: Fixed

Medium

No Medium severity vulnerabilities were found.

Low

(1) Function input parameters lack check:

Some functions require validation before execution.

Functions are:

- setPrice() - newPrice variable

- setBeneficiary() - _beneficiary variable

Resolution: We suggest using validation like for numerical variables that should be greater than 0 and for address type check variables that are not address(0).

Status: Acknowledged

Very Low / Informational / Best practices:

(1) Unused local libraries:

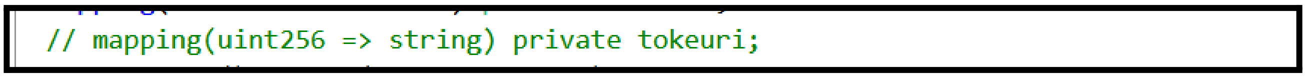
The Pausable library has been included in the contract code and the whenNotPaused() function is also used but the owner cannot set pause/unpause for the contract.

The Ownable library has been included in the contract code, but Ownable functions or modifiers are not used.

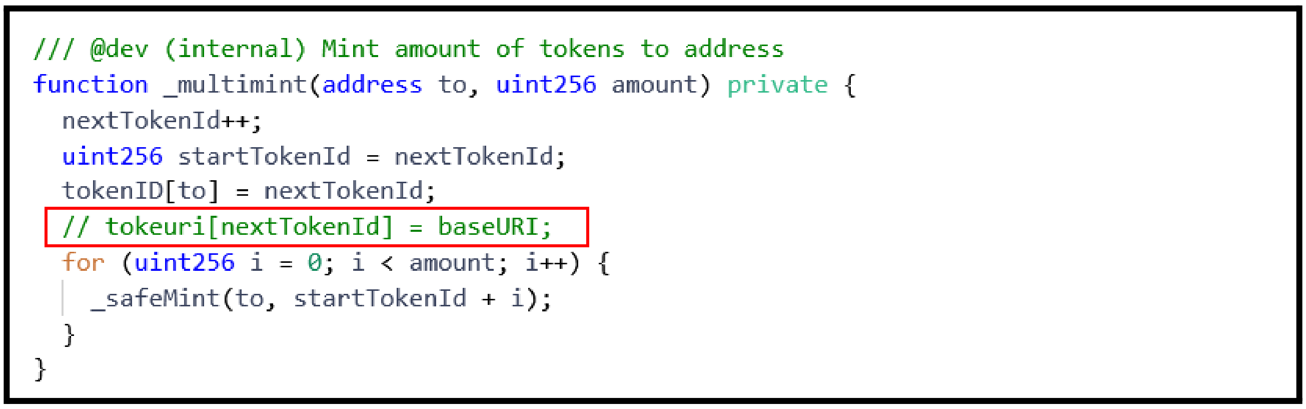
Resolution: We suggest removing unused libraries or adding owner functions to pause/unpause the contract.

Status: Acknowledged

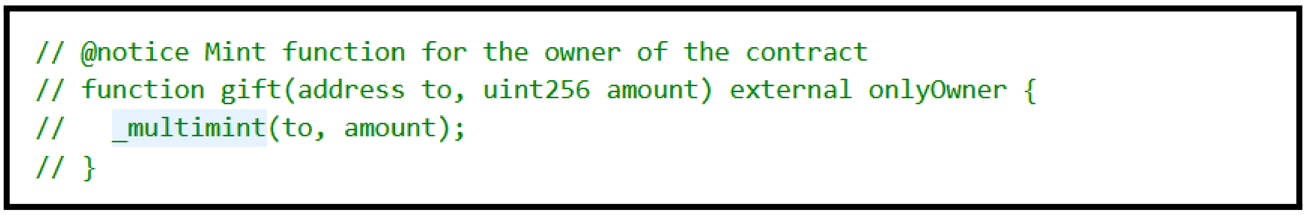
(2) Unwanted commented code:



```
// mapping(uint256 => string) private tokeuri;
```



```
/// @dev (internal) Mint amount of tokens to address
function _multimint(address to, uint256 amount) private {
  nextTokenId++;
  uint256 startTokenId = nextTokenId;
  tokenID[to] = nextTokenId;
  // tokeuri[nextTokenId] = baseURI;
  for (uint256 i = 0; i < amount; i++) {
    _safeMint(to, startTokenId + i);
  }
}
```



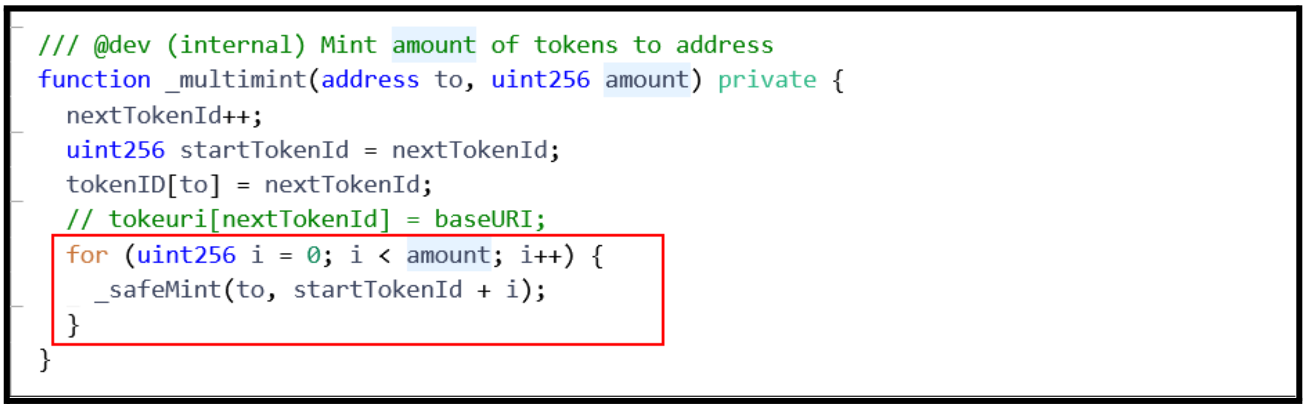
```
// @notice Mint function for the owner of the contract
// function gift(address to, uint256 amount) external onlyOwner {
//    _multimint(to, amount);
// }
```

There is an unwanted comment code added in the smart contract.

Resolution: We suggest removing unwanted commented codes.

Status: Fixed

(3) Unused for loop in mint tokens:



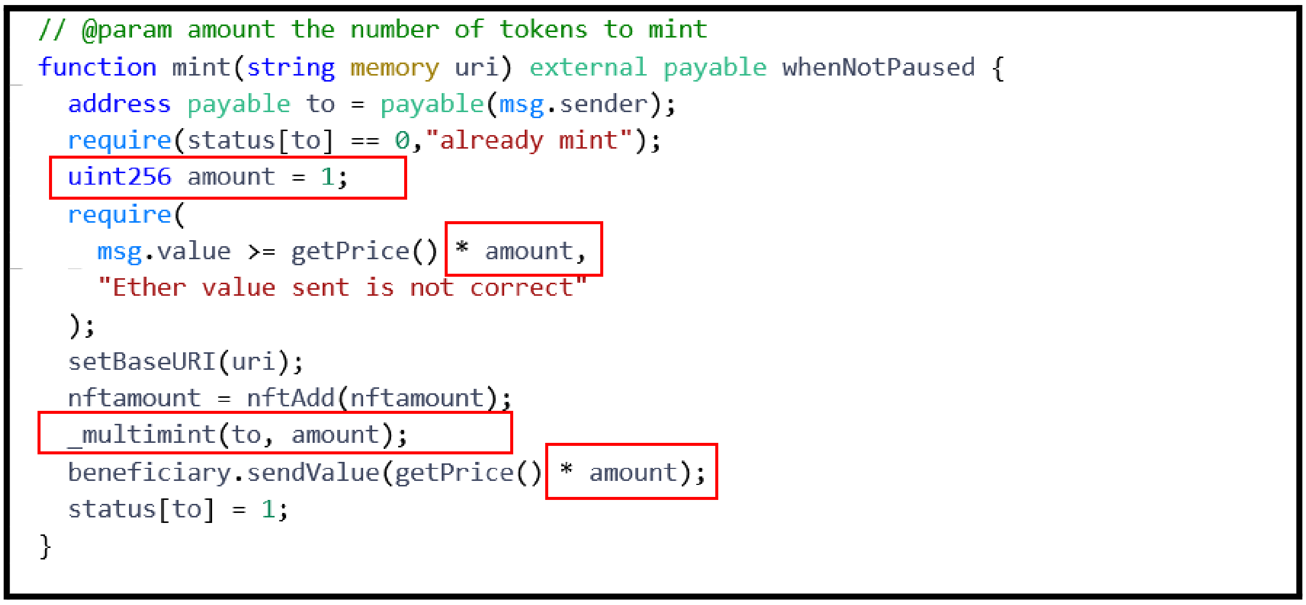
```
/// @dev (internal) Mint amount of tokens to address
function _multimint(address to, uint256 amount) private {
  nextTokenId++;
  uint256 startTokenId = nextTokenId;
  tokenID[to] = nextTokenId;
  // tokeuri[nextTokenId] = baseURI;
  for (uint256 i = 0; i < amount; i++) {
    _safeMint(to, startTokenId + i);
  }
}
```

The Mint function has an internal function _multimint() which is used to execute a loop for multiple token ID minting to the same address. but the Mint function is used to mint only 1 token at the time.

Resolution: We suggest removing the loop and execute the _safeMint() function while minting the token id.

Status: Fixed

(4) No need to define an amount variable for use in minting:



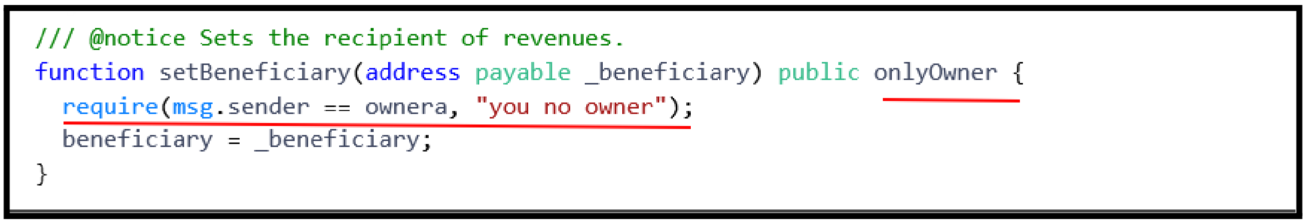
```
// @param amount the number of tokens to mint
function mint(string memory uri) external payable whenNotPaused {
  address payable to = payable(msg.sender);
  require(status[to] == 0,"already mint");
  uint256 amount = 1;
  require(
    msg.value >= getPrice() * amount,
    "Ether value sent is not correct"
  );
  setBaseURI(uri);
  nftamount = nftAdd(nftamount);
  _multimint(to, amount);
  beneficiary.sendValue(getPrice() * amount);
  status[to] = 1;
}
```

In the mint() function, the amount variable is set to static value 1 and used for further processing in minting.

Resolution: We suggest removing this variable and use a direct number for minting the token, It will save some gas.

Status: Fixed

(5) Checked both validation user is owner or not:



```
/// @notice Sets the recipient of revenues.
function setBeneficiary(address payable _beneficiary) public onlyOwner {
  require(msg.sender == ownera, "you no owner");
  beneficiary = _beneficiary;
}
```

In the setBeneficiary() function, these use the "onlyOwner" modifier and "require" to check if the caller user is the owner or not. So two times checking has been done here.

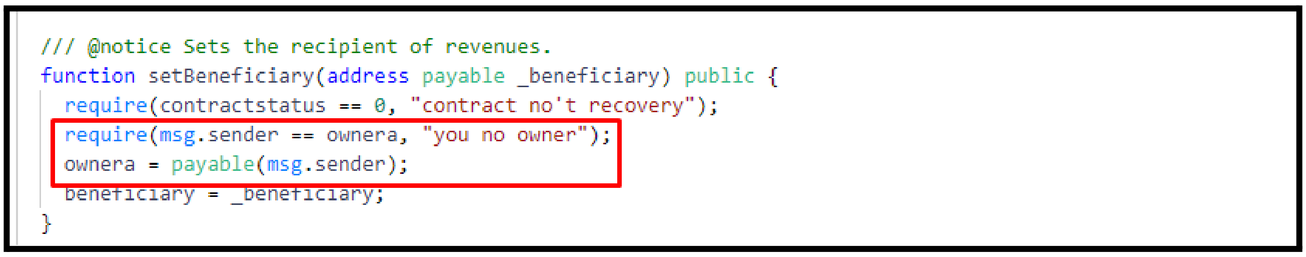
Resolution: We suggest removing this requirement to reduce some gas fee. Also remove "**ownera**" variable.

Status: Fixed

(6) Owner can stop the contract anytime:

There is a feature to enable/disable contact activity with contact status; the owner can enable or disable contract activity anytime.

(7) Owner is assigned the same address:



```
/// @notice Sets the recipient of revenues.
function setBeneficiary(address payable _beneficiary) public {
  require(contractstatus == 0, "contract no't recovery");
  require(msg.sender == ownera, "you no owner");
  ownera = payable(msg.sender);
  beneficiary = _beneficiary;
}
```

This function validates that the caller should be owner, and then again it sets the caller as owner. This is meaningless.

Resolution: We suggest removing the owner assignment by the same address to save some gas.

Centralization

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

- setBaseURI: The owner can set a new BaseURI.
- setBeneficiary: The owner can set the recipient of revenues.
- setPrice: The owner can set the new mint price.
- contractstop: The owner can stop the contract.
- contractrecovery: The owner can restart the contract.

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 file And we have used all possible tests based on given objects as files. We have observed some issues in the smart contracts and those issues have been resolved/acknowledged in the revised code. **So, the smart contracts are ready for the mainnet deployment**.

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 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.

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

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 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 whitebox penetration testing. We look at the project's web site 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).

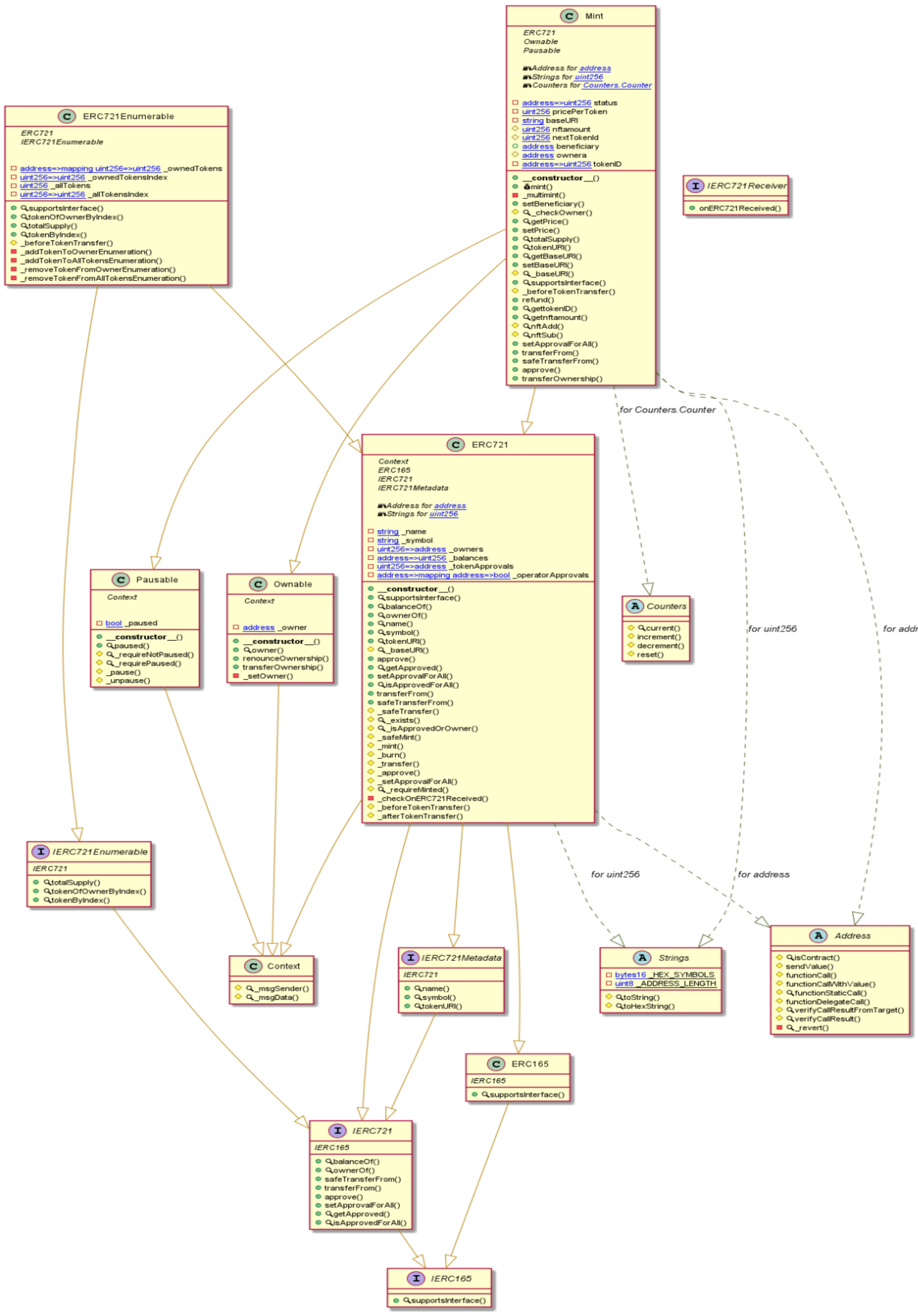
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While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

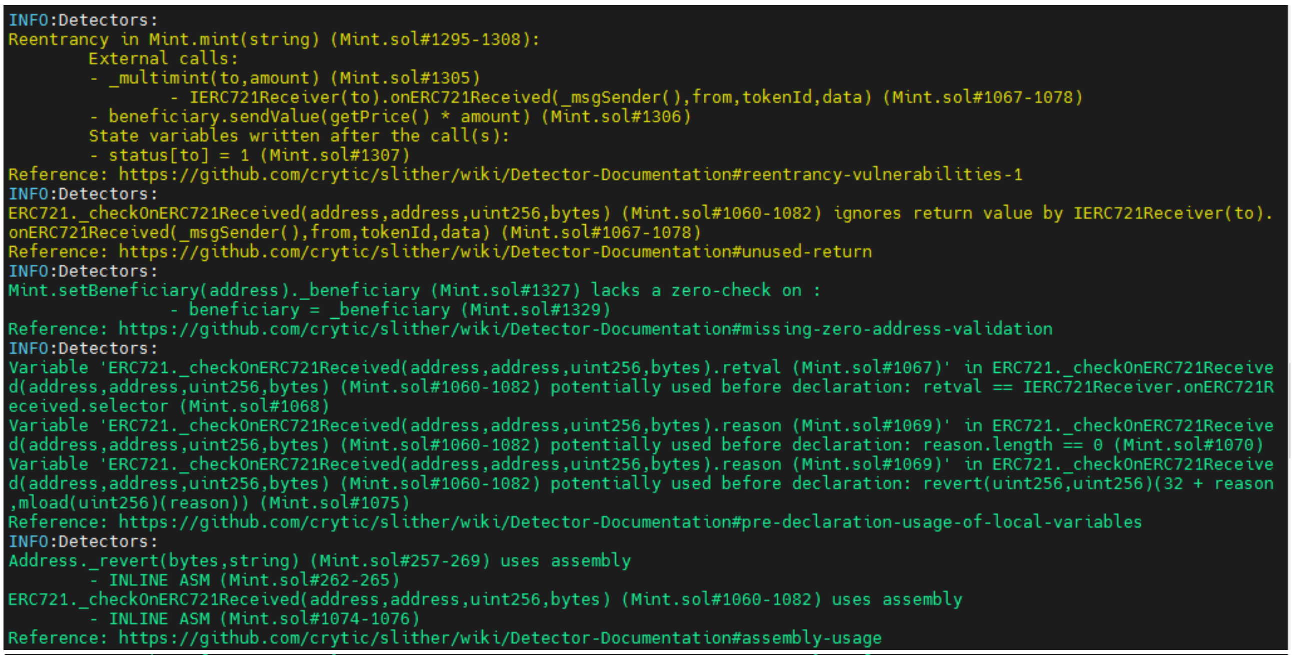
Appendix

Code Flow Diagram - Key4 Token

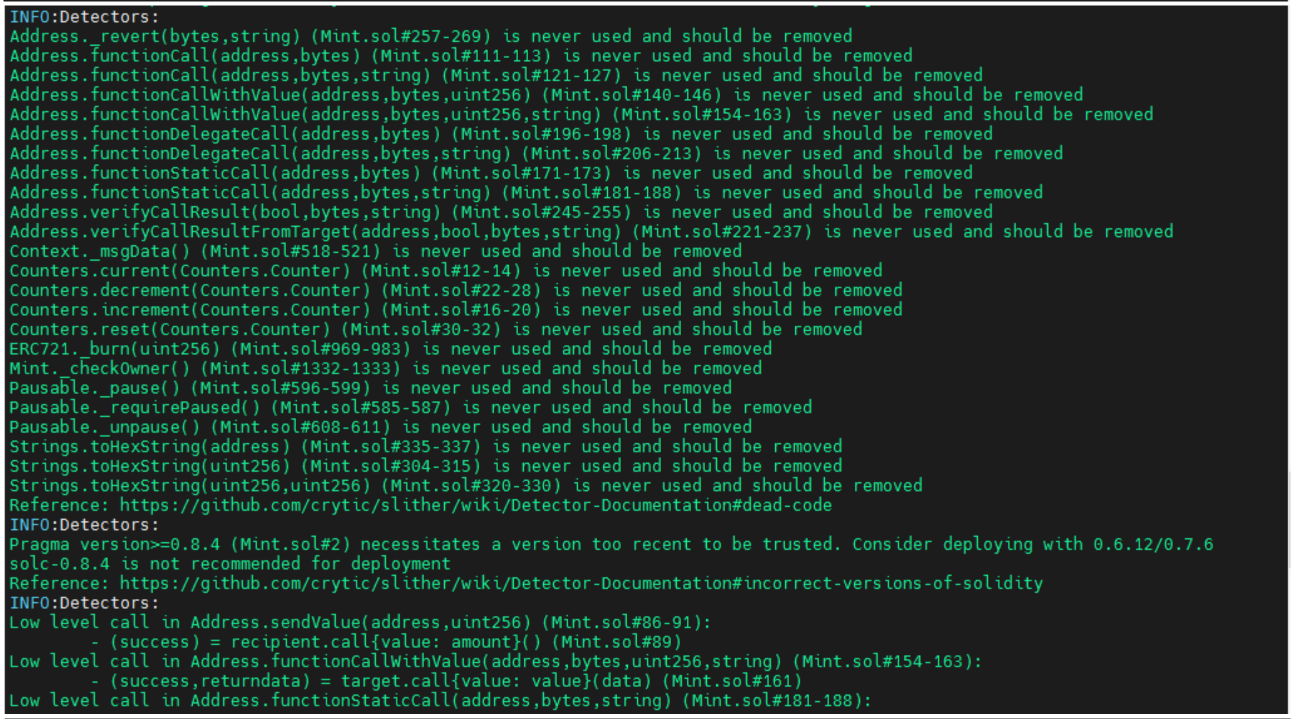


Slither Results Log

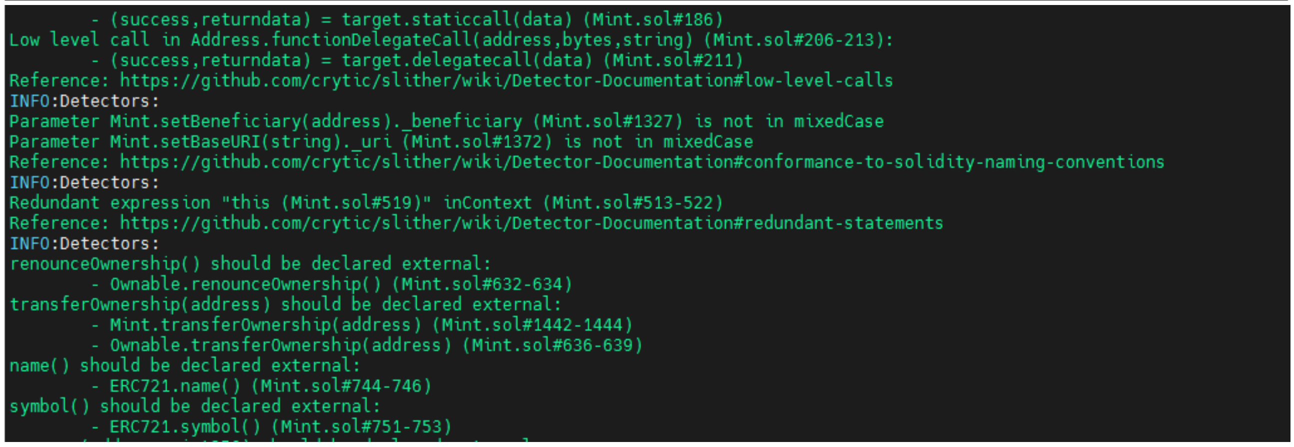
Slither Log >> Mint.sol



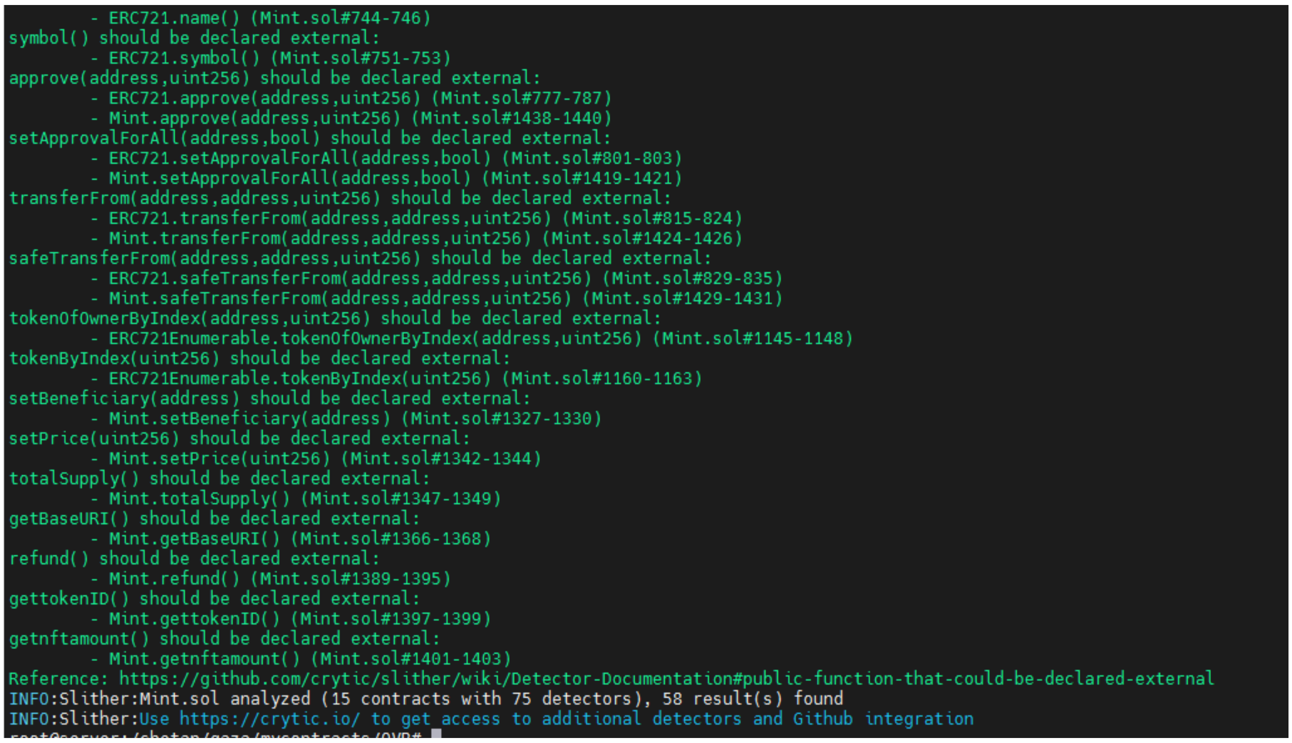
```
INFO:Detectors:
Reentrancy in Mint.mint(string) (Mint.sol#1295-1308):
        External calls:
        - _multimint(to,amount) (Mint.sol#1305)
                - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (Mint.sol#1067-1078)
        - beneficiary.sendValue(getPrice() * amount) (Mint.sol#1306)
        State variables written after the call(s):
        - status[to] = 1 (Mint.sol#1307)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
ERC721._checkOnERC721Received(address,address,uint256,bytes) (Mint.sol#1060-1082) ignores return value by IERC721Receiver(to).
onERC721Received(_msgSender(),from,tokenId,data) (Mint.sol#1067-1078)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Mint.setBeneficiary(address)._beneficiary (Mint.sol#1327) lacks a zero-check on :
                - beneficiary = _beneficiary (Mint.sol#1329)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).retval (Mint.sol#1067)' in ERC721._checkOnERC721Receive
d(address,address,uint256,bytes) (Mint.sol#1060-1082) potentially used before declaration: retval == IERC721Receiver.onERC721R
eceived.selector (Mint.sol#1068)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (Mint.sol#1069)' in ERC721._checkOnERC721Receive
d(address,address,uint256,bytes) (Mint.sol#1060-1082) potentially used before declaration: reason.length == 0 (Mint.sol#1070)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (Mint.sol#1069)' in ERC721._checkOnERC721Receive
d(address,address,uint256,bytes) (Mint.sol#1060-1082) potentially used before declaration: revert(uint256,uint256)(32 + reason
,mload(uint256)(reason)) (Mint.sol#1075)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
INFO:Detectors:
Address._revert(bytes,string) (Mint.sol#257-269) uses assembly
        - INLINE ASM (Mint.sol#262-265)
ERC721._checkOnERC721Received(address,address,uint256,bytes) (Mint.sol#1060-1082) uses assembly
        - INLINE ASM (Mint.sol#1074-1076)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```



```
INFO:Detectors:
Address._revert(bytes,string) (Mint.sol#257-269) is never used and should be removed
Address.functionCall(address,bytes) (Mint.sol#111-113) is never used and should be removed
Address.functionCall(address,bytes,string) (Mint.sol#121-127) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Mint.sol#140-146) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (Mint.sol#154-163) is never used and should be removed
Address.functionDelegateCall(address,bytes) (Mint.sol#196-198) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (Mint.sol#206-213) is never used and should be removed
Address.functionStaticCall(address,bytes) (Mint.sol#171-173) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (Mint.sol#181-188) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (Mint.sol#245-255) is never used and should be removed
Address.verifyCallResultFromTarget(address,bool,bytes,string) (Mint.sol#221-237) is never used and should be removed
Context._msgData() (Mint.sol#518-521) is never used and should be removed
Counters.current(Counters.Counter) (Mint.sol#12-14) is never used and should be removed
Counters.decrement(Counters.Counter) (Mint.sol#22-28) is never used and should be removed
Counters.increment(Counters.Counter) (Mint.sol#16-20) is never used and should be removed
Counters.reset(Counters.Counter) (Mint.sol#30-32) is never used and should be removed
ERC721._burn(uint256) (Mint.sol#969-983) is never used and should be removed
Mint._checkOwner() (Mint.sol#1332-1333) is never used and should be removed
Pausable._pause() (Mint.sol#596-599) is never used and should be removed
Pausable._requirePaused() (Mint.sol#585-587) is never used and should be removed
Pausable._unpause() (Mint.sol#608-611) is never used and should be removed
Strings.toHexString(address) (Mint.sol#335-337) is never used and should be removed
Strings.toHexString(uint256) (Mint.sol#304-315) is never used and should be removed
Strings.toHexString(uint256,uint256) (Mint.sol#320-330) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version>=0.8.4 (Mint.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 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) (Mint.sol#86-91):
        - (success) = recipient.call{value: amount}() (Mint.sol#89)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (Mint.sol#154-163):
        - (success,returndata) = target.call{value: value}(data) (Mint.sol#161)
Low level call in Address.functionStaticCall(address,bytes,string) (Mint.sol#181-188):
```



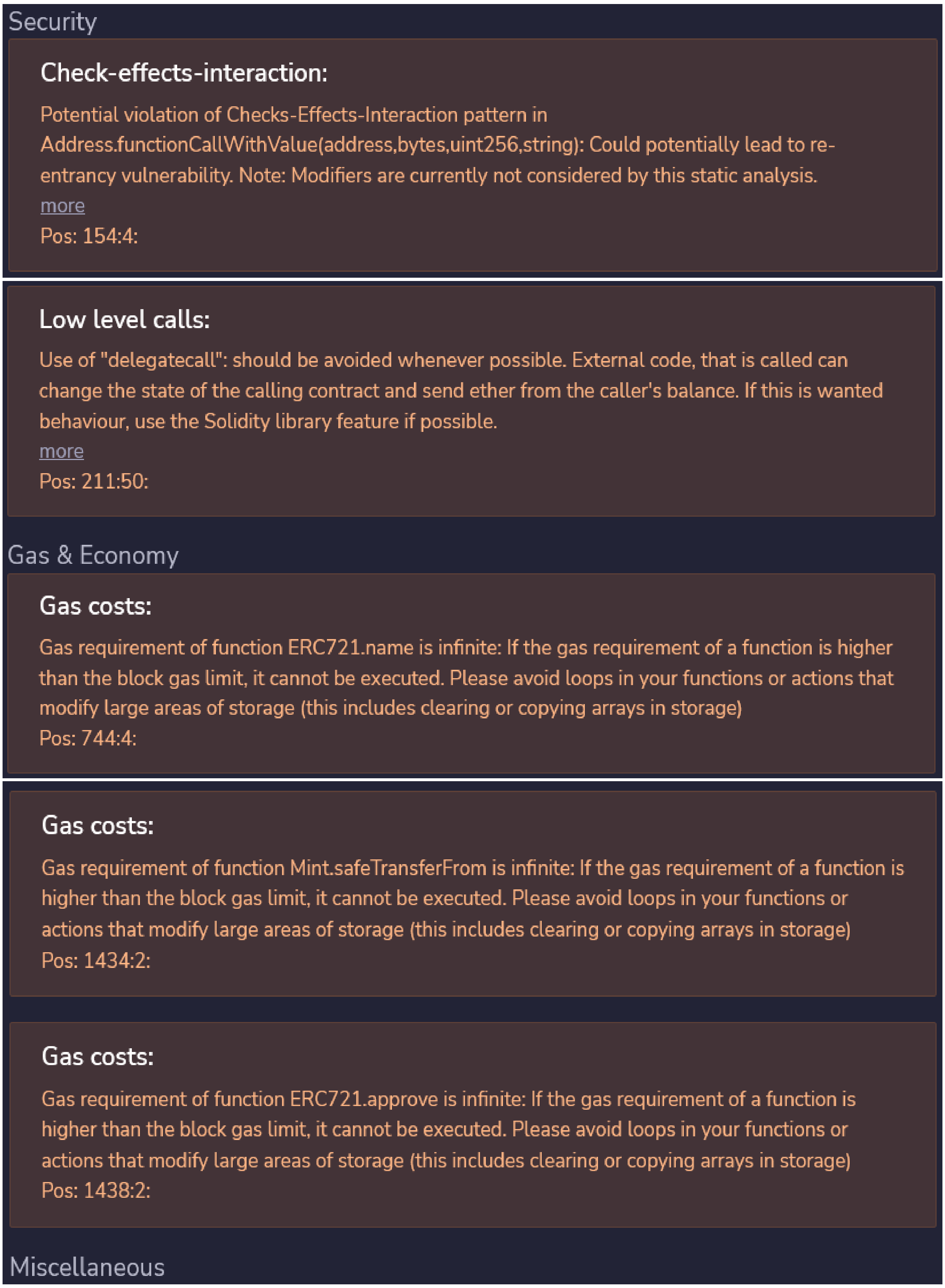
```
        - (success,returndata) = target.staticcall(data) (Mint.sol#186)
Low level call in Address.functionDelegateCall(address,bytes,string) (Mint.sol#206-213):
        - (success,returndata) = target.delegatecall(data) (Mint.sol#211)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter Mint.setBeneficiary(address)._beneficiary (Mint.sol#1327) is not in mixedCase
Parameter Mint.setBaseURI(string)._uri (Mint.sol#1372) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (Mint.sol#519)" inContext (Mint.sol#513-522)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (Mint.sol#632-634)
transferOwnership(address) should be declared external:
        - Mint.transferOwnership(address) (Mint.sol#1442-1444)
        - Ownable.transferOwnership(address) (Mint.sol#636-639)
name() should be declared external:
        - ERC721.name() (Mint.sol#744-746)
symbol() should be declared external:
        - ERC721.symbol() (Mint.sol#751-753)
```



```
        - ERC721.name() (Mint.sol#744-746)
symbol() should be declared external:
        - ERC721.symbol() (Mint.sol#751-753)
approve(address,uint256) should be declared external:
        - ERC721.approve(address,uint256) (Mint.sol#777-787)
        - Mint.approve(address,uint256) (Mint.sol#1438-1440)
setApprovalForAll(address,bool) should be declared external:
        - ERC721.setApprovalForAll(address,bool) (Mint.sol#801-803)
        - Mint.setApprovalForAll(address,bool) (Mint.sol#1419-1421)
transferFrom(address,address,uint256) should be declared external:
        - ERC721.transferFrom(address,address,uint256) (Mint.sol#815-824)
        - Mint.transferFrom(address,address,uint256) (Mint.sol#1424-1426)
safeTransferFrom(address,address,uint256) should be declared external:
        - ERC721.safeTransferFrom(address,address,uint256) (Mint.sol#829-835)
        - Mint.safeTransferFrom(address,address,uint256) (Mint.sol#1429-1431)
tokenOfOwnerByIndex(address,uint256) should be declared external:
        - ERC721Enumerable.tokenOfOwnerByIndex(address,uint256) (Mint.sol#1145-1148)
tokenByIndex(uint256) should be declared external:
        - ERC721Enumerable.tokenByIndex(uint256) (Mint.sol#1160-1163)
setBeneficiary(address) should be declared external:
        - Mint.setBeneficiary(address) (Mint.sol#1327-1330)
setPrice(uint256) should be declared external:
        - Mint.setPrice(uint256) (Mint.sol#1342-1344)
totalSupply() should be declared external:
        - Mint.totalSupply() (Mint.sol#1347-1349)
getBaseURI() should be declared external:
        - Mint.getBaseURI() (Mint.sol#1366-1368)
refund() should be declared external:
        - Mint.refund() (Mint.sol#1389-1395)
gettokenID() should be declared external:
        - Mint.gettokenID() (Mint.sol#1397-1399)
getnftamount() should be declared external:
        - Mint.getnftamount() (Mint.sol#1401-1403)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Mint.sol analyzed (15 contracts with 75 detectors), 58 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

Mint.sol



Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Address.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 154:4:

Low level calls:

Use of "delegatecall": should be avoided whenever possible. External code, that is called can change the state of the calling contract and send ether from the caller's balance. If this is wanted behaviour, use the Solidity library feature if possible.
more
Pos: 211:50:

Gas & Economy

Gas costs:

Gas requirement of function ERC721.name 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: 744:4:

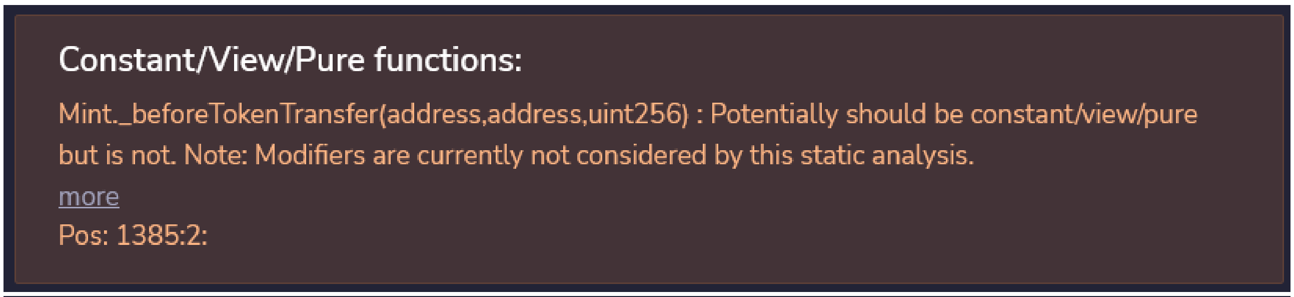
Gas costs:

Gas requirement of function Mint.safeTransferFrom 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: 1434:2:

Gas costs:

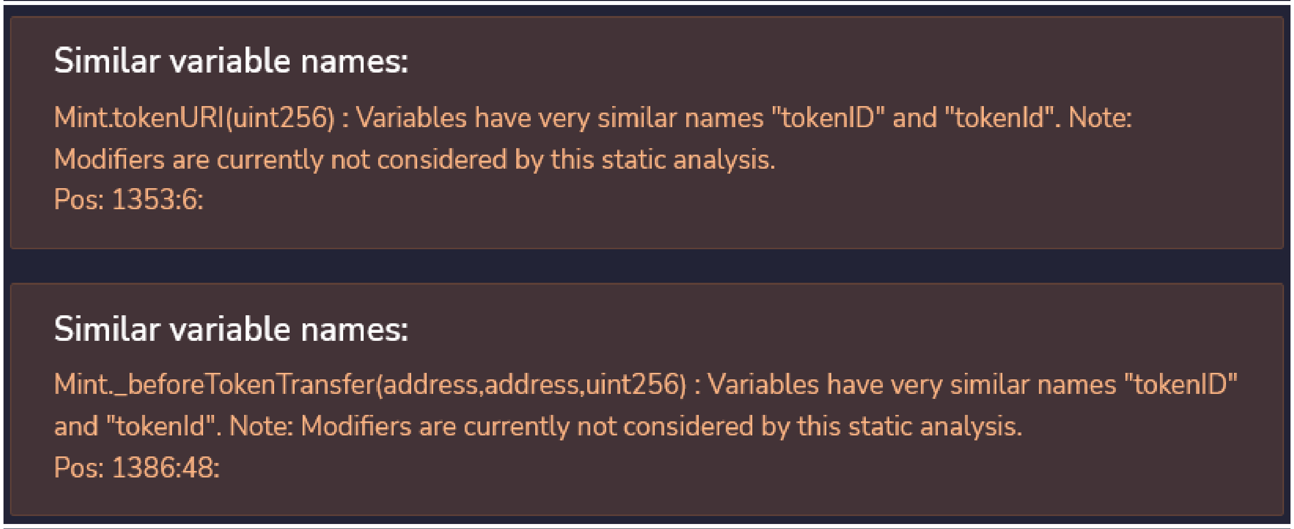
Gas requirement of function ERC721.approve is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1438:2:

Miscellaneous



Constant/View/Pure functions:

Mint._beforeTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1385:2:

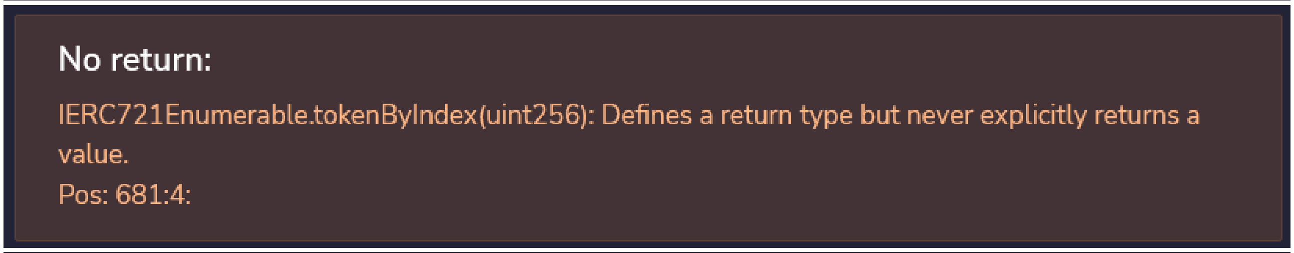


Similar variable names:

Mint.tokenURI(uint256) : Variables have very similar names "tokenID" and "tokenId". Note: Modifiers are currently not considered by this static analysis.
Pos: 1353:6:

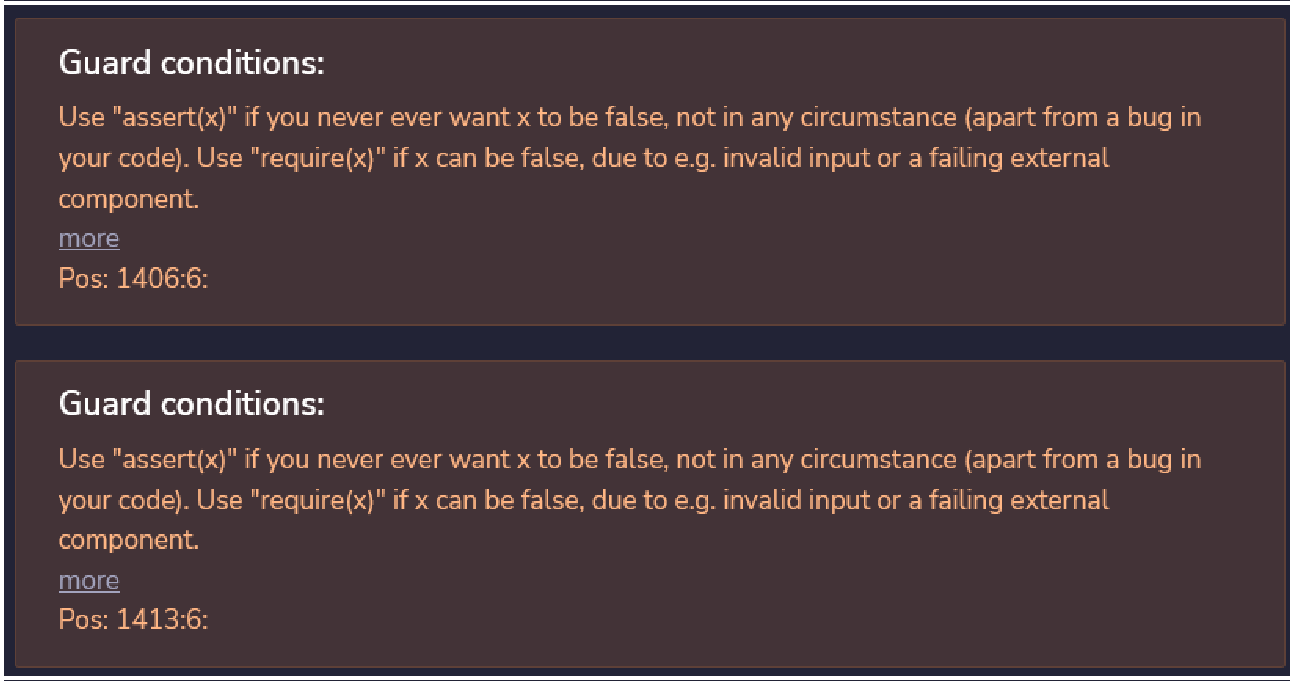
Similar variable names:

Mint._beforeTokenTransfer(address,address,uint256) : Variables have very similar names "tokenID" and "tokenId". Note: Modifiers are currently not considered by this static analysis.
Pos: 1386:48:



No return:

IERC721Enumerable.tokenByIndex(uint256): Defines a return type but never explicitly returns a value.
Pos: 681:4:

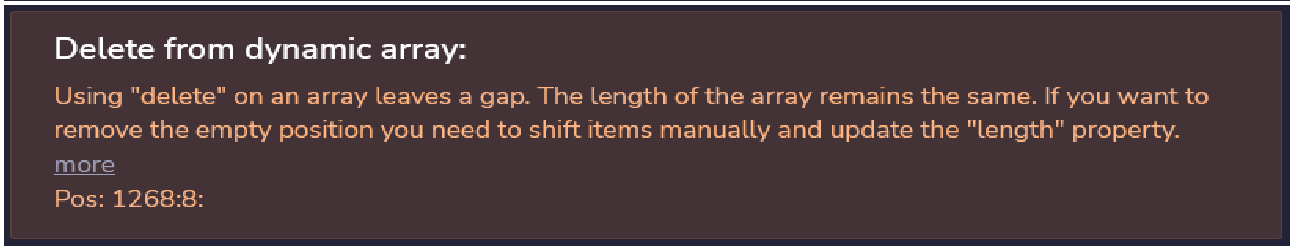


Guard conditions:

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: 1406:6:

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: 1413:6:

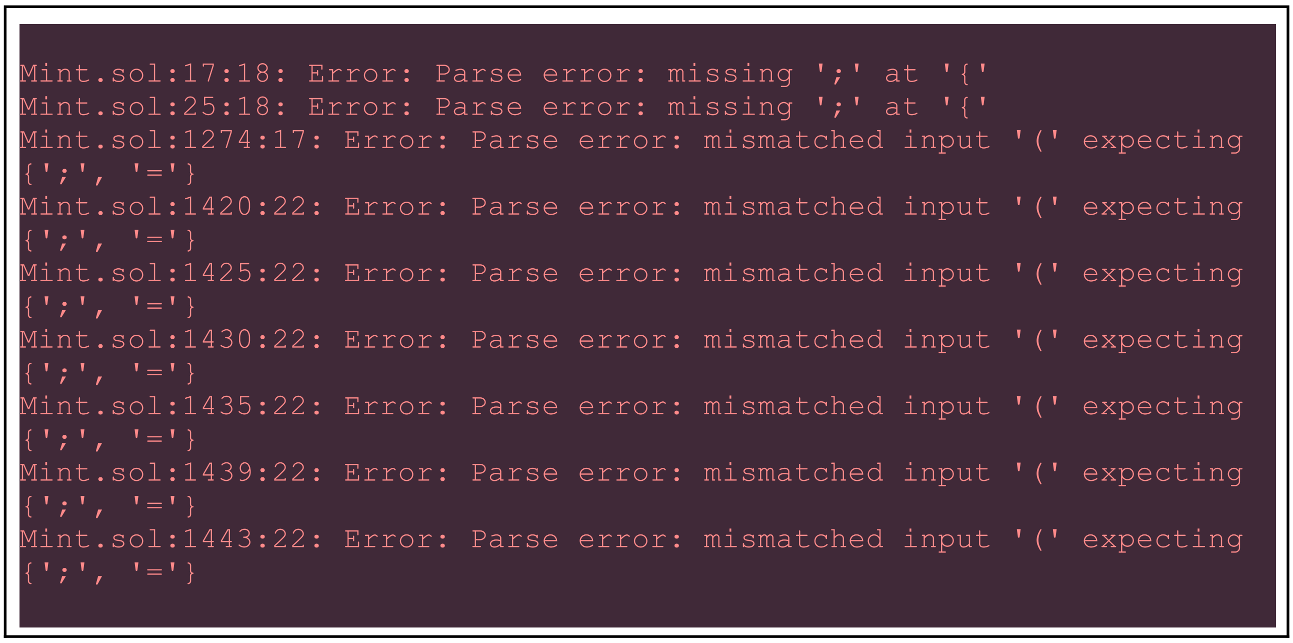


Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
more
Pos: 1268:8:

Solhint Linter

Mint.sol



```
Mint.sol:17:18: Error: Parse error: missing ';' at '{'
Mint.sol:25:18: Error: Parse error: missing ';' at '{'
Mint.sol:1274:17: Error: Parse error: mismatched input '(' expecting
{';', '='}
Mint.sol:1420:22: Error: Parse error: mismatched input '(' expecting
{';', '='}
Mint.sol:1425:22: Error: Parse error: mismatched input '(' expecting
{';', '='}
Mint.sol:1430:22: Error: Parse error: mismatched input '(' expecting
{';', '='}
Mint.sol:1435:22: Error: Parse error: mismatched input '(' expecting
{';', '='}
Mint.sol:1439:22: Error: Parse error: mismatched input '(' expecting
{';', '='}
Mint.sol:1443:22: Error: Parse error: mismatched input '(' expecting
{';', '='}
```

Software analysis result:

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

EtherAuthority