its___msn itsmsn

ITISH

AUDIT COMPANY

Audit Details

Contract Name

VirulentApesTownClub

Deployer address

0xc1f46ef393e228e96fc1e8a250cdb0b19db08f5d

Client contacts:

VirulentApesTownClub

Blockchain:

Ethereum

Project website:

Not provided by client

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Itish and its affiliates (includingholding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Itish) owe no duty of care towards you or any other person, nor does Itish make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Itish hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Itish hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Itish, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

Background

Itish was commissioned Virulentapestownclub Contract to perform an audit of smart contracts:

https://etherscan.io/address/0xc1f46ef393e228e96fc1e8a250cdb0b19db08f5d

The purpose of the audit was to achieve the following:

- **Ensure that the smart contract functions as intended.**
- **Identify potential security issues with the smart contract.**

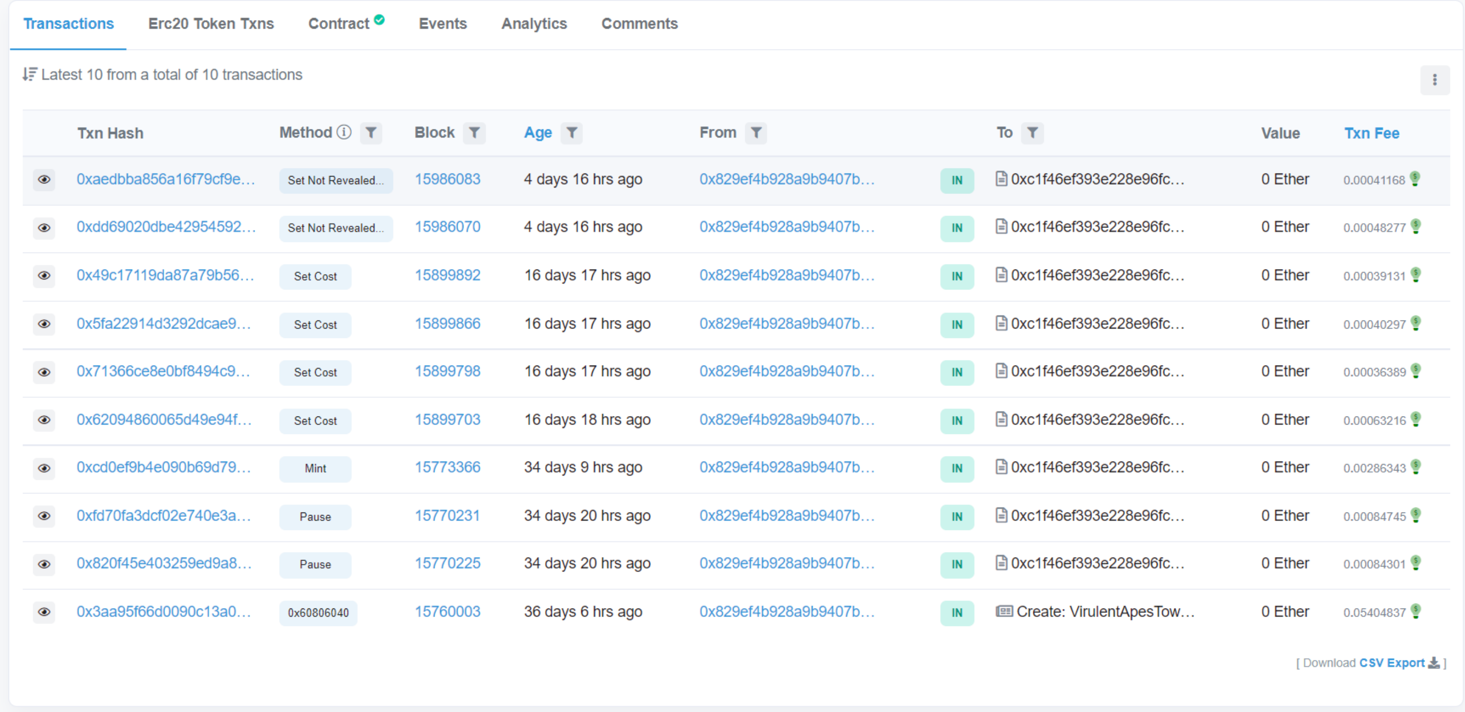
The information in this report should be used to understand the risk exposure of the smart contract, and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

Contract Details

Token contract details for 21.11.2022

contract name	**Virulentapestownclub**
Contract creator	0xc1f46ef393e228e96fc1e8a250cdb0b19db08f5d
Transaction's count	**10**

Contract TopTransactions



Latest 10 from a total of 10 transactions

	Txn Hash	Method	Block	Age	From		To	Value	Txn Fee
👁	0xaedbba856a16f79cf9e…	Set Not Revealed…	15986083	4 days 16 hrs ago	0x829ef4b928a9b9407b…	IN	0xc1f46ef393e228e96fc…	0 Ether	0.00041168
👁	0xdd69020dbe42954592…	Set Not Revealed…	15986070	4 days 16 hrs ago	0x829ef4b928a9b9407b…	IN	0xc1f46ef393e228e96fc…	0 Ether	0.00048277
👁	0x49c17119da87a79b56…	Set Cost	15899892	16 days 17 hrs ago	0x829ef4b928a9b9407b…	IN	0xc1f46ef393e228e96fc…	0 Ether	0.00039131
👁	0x5fa22914d3292dcae9…	Set Cost	15899866	16 days 17 hrs ago	0x829ef4b928a9b9407b…	IN	0xc1f46ef393e228e96fc…	0 Ether	0.00040297
👁	0x71366ce8e0bf8494c9…	Set Cost	15899798	16 days 17 hrs ago	0x829ef4b928a9b9407b…	IN	0xc1f46ef393e228e96fc…	0 Ether	0.00036389
👁	0x62094860065d49e94f…	Set Cost	15899703	16 days 18 hrs ago	0x829ef4b928a9b9407b…	IN	0xc1f46ef393e228e96fc…	0 Ether	0.00063216
👁	0xcd0ef9b4e090b69d79…	Mint	15773366	34 days 9 hrs ago	0x829ef4b928a9b9407b…	IN	0xc1f46ef393e228e96fc…	0 Ether	0.00286343
👁	0xfd70fa3dcf02e740e3a…	Pause	15770231	34 days 20 hrs ago	0x829ef4b928a9b9407b…	IN	0xc1f46ef393e228e96fc…	0 Ether	0.00084745
👁	0x820f45e403259ed9a8…	Pause	15770225	34 days 20 hrs ago	0x829ef4b928a9b9407b…	IN	0xc1f46ef393e228e96fc…	0 Ether	0.00084301
👁	0x3aa95f66d0090c13a0…	0x60806040	15760003	36 days 6 hrs ago	0x829ef4b928a9b9407b…	IN	Create: VirulentApesTow…	0 Ether	0.05404837

[Download **CSV Export**]

Token Functions Details

Addressminted
Balanceof
baseExtension
Baseuri
cost
getApproved
isApprovedFor
isWhitelisted
maxSupply
name
owner
ownerof
paused
revealed
symbol
tokenByIndex
Totalsupply
walletofowner

Contract Interface Details

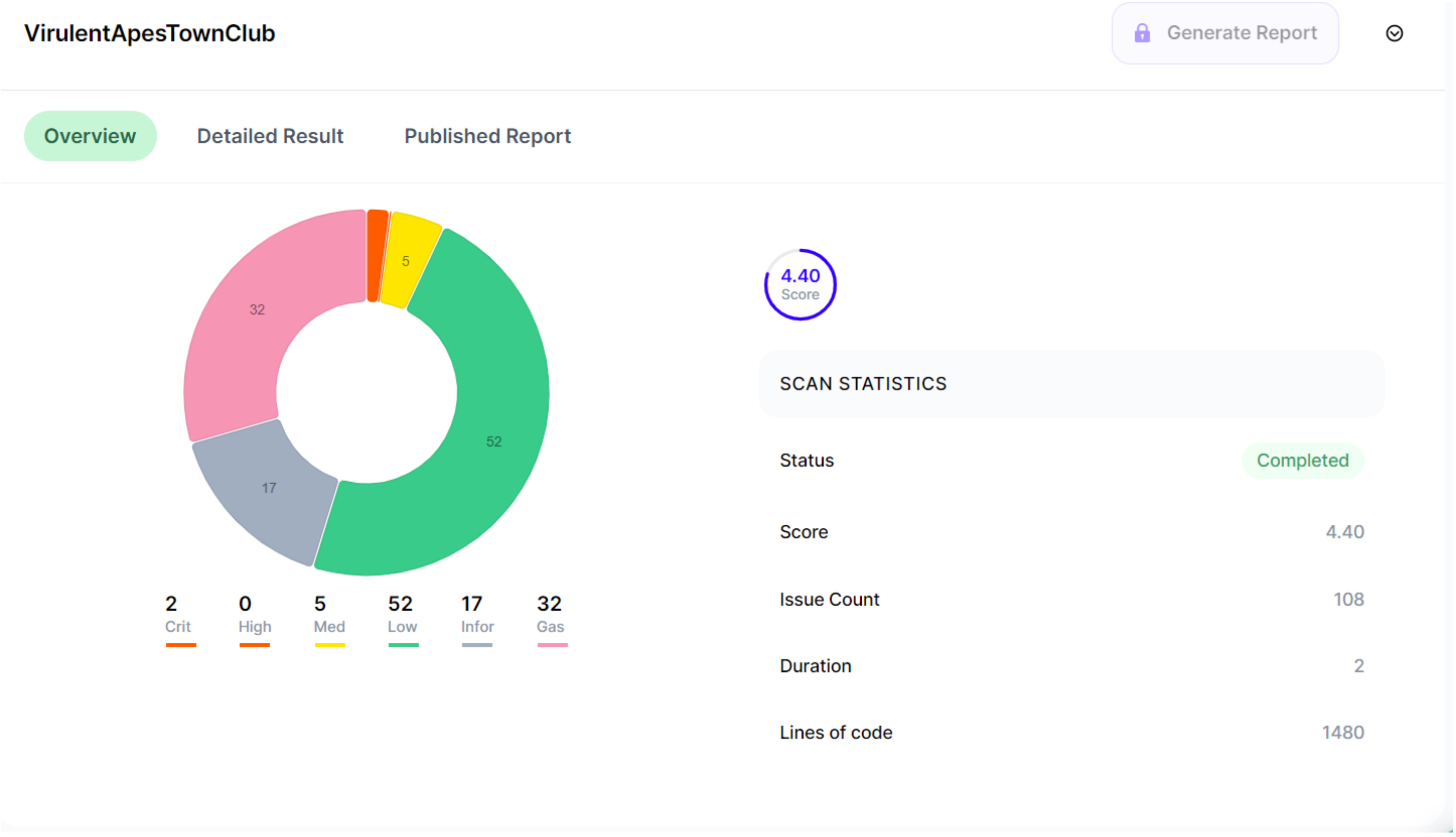
```
interface IERC20
interface IERC20Metadata is IERC20
```

Issues Checking Status

	Issue description	Checking status
1.	Compiler errors.	Passed
2.	Compiler Compatibilities	Failed
3.	Possible delays in data delivery.	Passed
4.	Oracle calls.	Moderate
5.	Front running.	Failed
6.	Timestamp dependence.	Passed
7.	Integer Overflow and Underflow.	Passed
8.	DoS with Revert.	Severe
9.	DoS with block gas limit.	Moderate
10.	Methods execution permissions.	Passed
11.	Economy model of the contract.	Passed
12.	The impact of the exchange rate on the logic.	Severe
13.	Private user data leaks.	Passed
14.	Malicious Event log.	Passed
15.	Scoping and Declarations.	Passed
16.	Uninitialized storage pointers.	Passed
17.	Arithmetic accuracy.	Moderate
18.	Design Logic.	Moderate

19.	**Cross-function race conditions.**	Passed
20 .	Safe Open Zeppelin contracts implementation and usage.	Failed
21.	**Fallback function security.**	Failed

Security Issues



SCAN STATISTICS

Status	Completed
Score	4.40
Issue Count	108
Duration	2
Lines of code	1480

Critical Security Issues

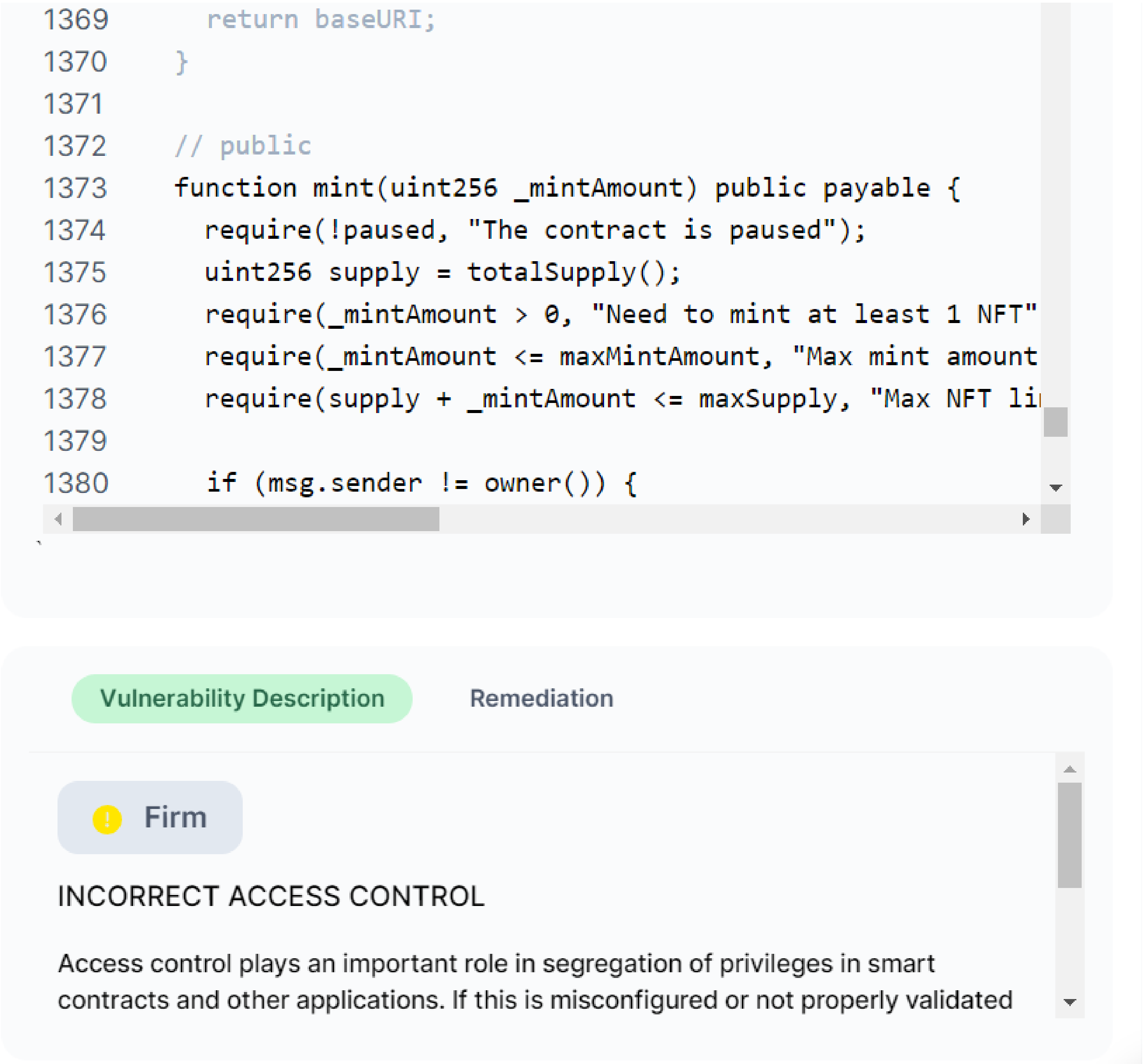
Critical security issues found

Issue # 1:

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract `VirulentApesTownClub` is importing an access control library `@openzeppelin/contracts/access/Ownable.sol` but the function `mint` is missing the modifier `onlyOwner`..



```
      return baseURI;
    }

    // public
    function mint(uint256 _mintAmount) public payable {
      require(!paused, "The contract is paused");
      uint256 supply = totalSupply();
      require(_mintAmount > 0, "Need to mint at least 1 NFT"
      require(_mintAmount <= maxMintAmount, "Max mint amount
      require(supply + _mintAmount <= maxSupply, "Max NFT li

      if (msg.sender != owner()) {
```

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated

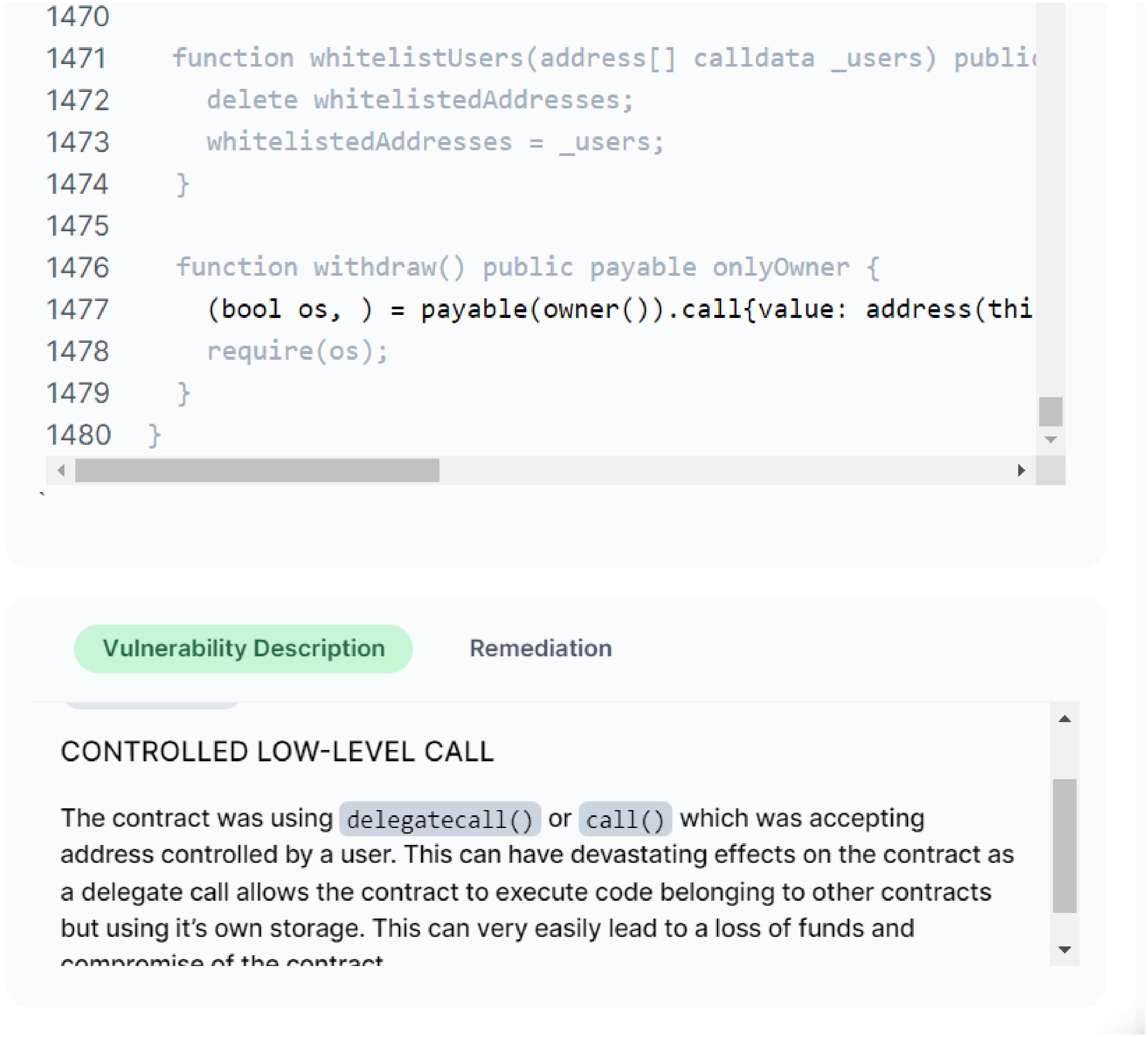
Remediation # 1:

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same.

Issue # 2:

CONTROLLED LOW-LEVEL CALL

The contract was using `delegatecall()` or `call()` which was accepting address controlled by a user. This can have devastating effects on the contract as a delegate call allows the contract to execute code belonging to other contracts but using it’s own storage. This can very easily lead to a loss of funds and compromise of the contract.



```

  function whitelistUsers(address[] calldata _users) publi
    delete whitelistedAddresses;
    whitelistedAddresses = _users;
  }

  function withdraw() public payable onlyOwner {
    (bool os, ) = payable(owner()).call{value: address(thi
    require(os);
  }
}
```

CONTROLLED LOW-LEVEL CALL

The contract was using `delegatecall()` or `call()` which was accepting address controlled by a user. This can have devastating effects on the contract as a delegate call allows the contract to execute code belonging to other contracts but using it's own storage. This can very easily lead to a loss of funds and compromise of the contract

Remediation # 2:

Do not allow user-controlled data inside the `delegatecall()` and the `call()` function.

High Severity Issues

NO High security issues found

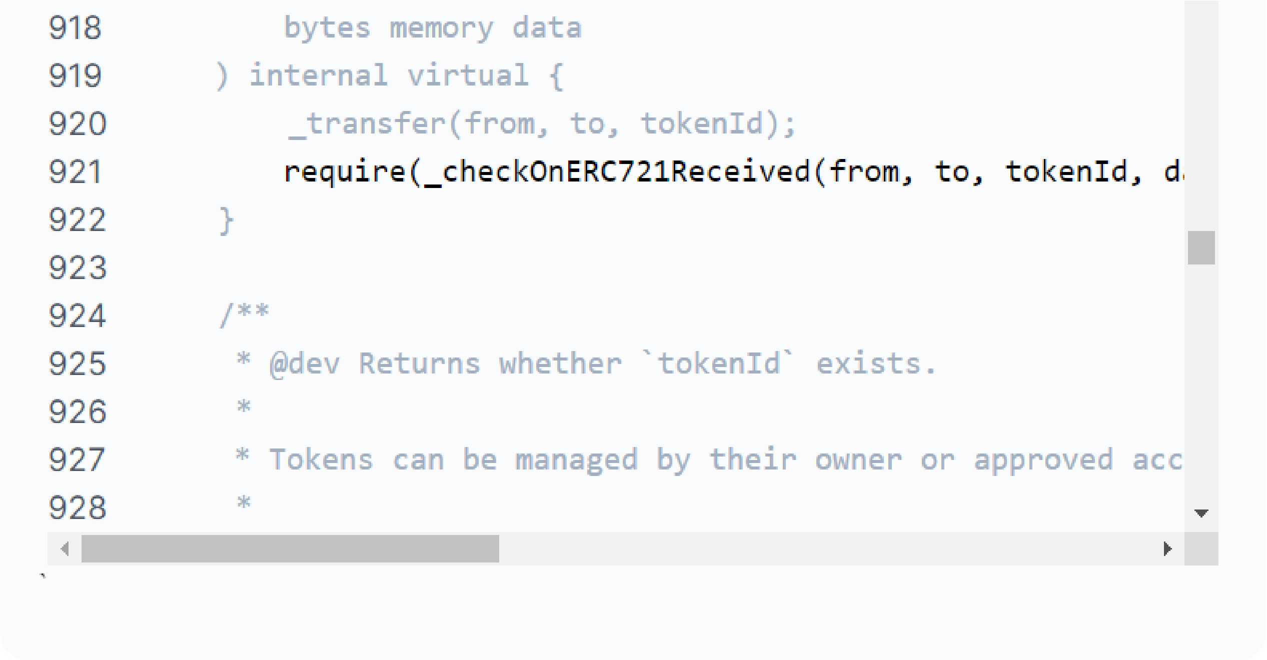
Medium Severity Issues

Issue # 1:

ASSERT REQUIRE STATE CHANGES:

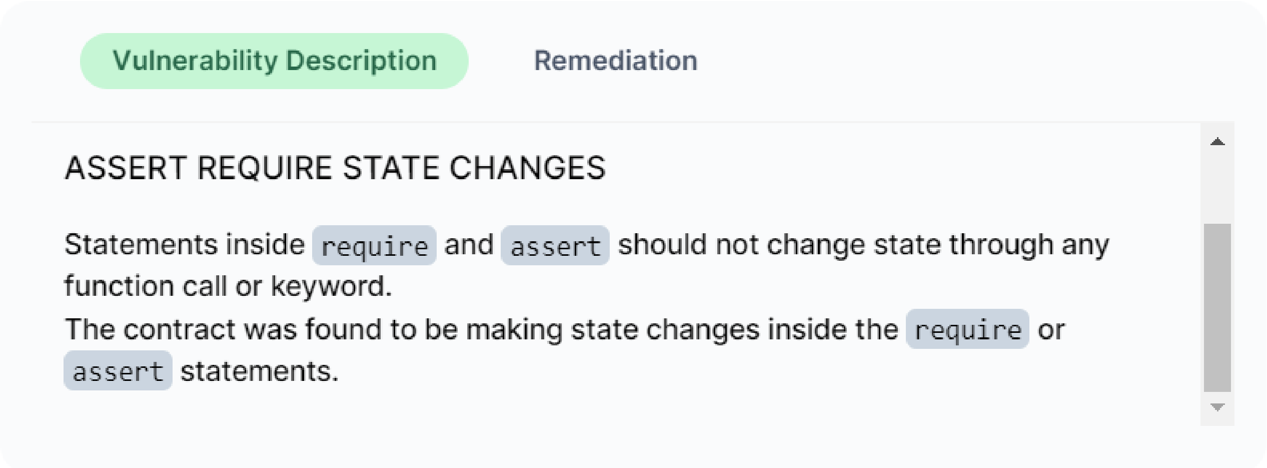
Statements inside `require` and `assert` should not change state through any function call or keyword.

The contract was found to be making state changes inside the `require` or `assert` statements.



```
        bytes memory data
    ) internal virtual {
        _transfer(from, to, tokenId);
        require(_checkOnERC721Received(from, to, tokenId, d
    }

    /**
     * @dev Returns whether `tokenId` exists.
     *
     * Tokens can be managed by their owner or approved acc
     *
```



ASSERT REQUIRE STATE CHANGES

Statements inside `require` and `assert` should not change state through any function call or keyword.

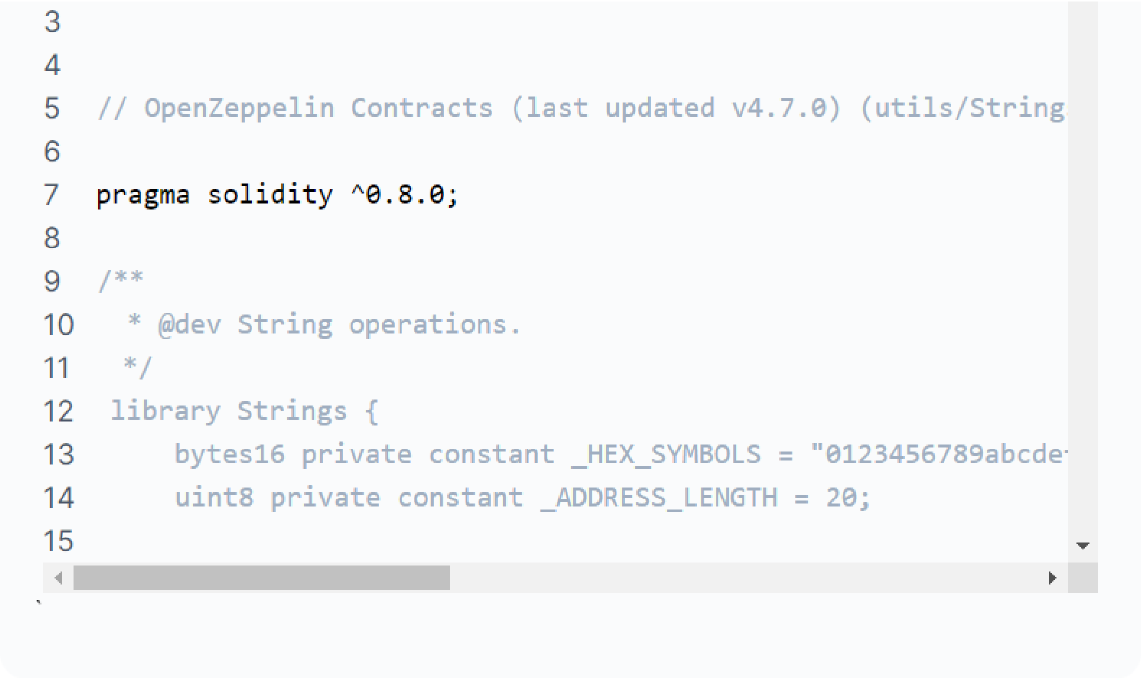
The contract was found to be making state changes inside the `require` or `assert` statements.

Remediation # 1:

It is recommended to not make any state changes inside `assert` or `require` statements and to always follow the pattern of check-effects-interactions.
`assert` should only be used to check invariants and should be replaced with `require` for user input and return values.

Issue # 2:

USE OF FLOATING PRAGMA:
Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the top of the solidity file. This can either be a floating pragma or a specific compiler version.
The contract was found to be using a floating pragma which is not considered safe as it can be compiled with all the versions described.
The following affected files were found to be using floating pragma:
`contract.sol - ^0.8.0`



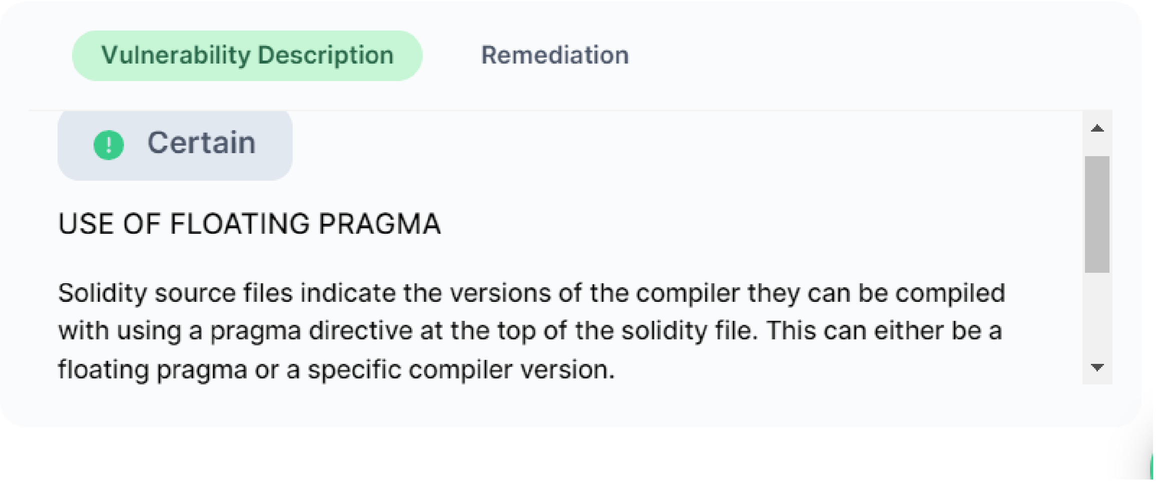
```


// OpenZeppelin Contracts (last updated v4.7.0) (utils/String

pragma solidity ^0.8.0;

/**
  * @dev String operations.
  */
 library Strings {
     bytes16 private constant _HEX_SYMBOLS = "0123456789abcde
     uint8 private constant _ADDRESS_LENGTH = 20;

```



USE OF FLOATING PRAGMA

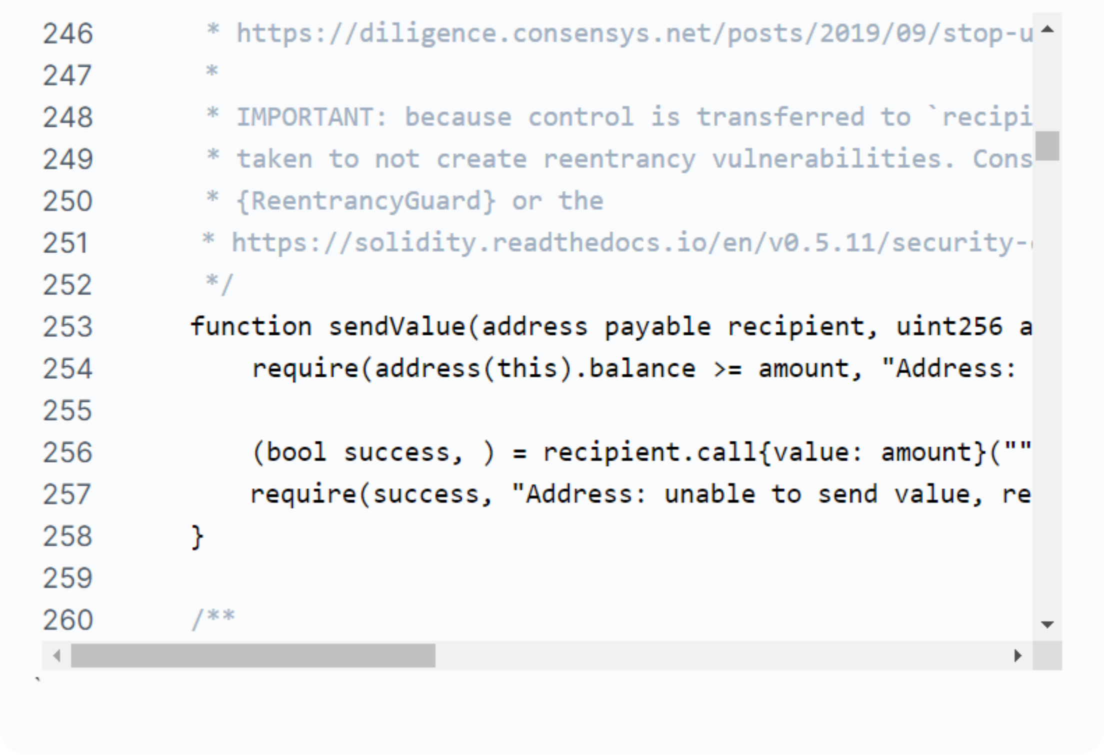
Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the top of the solidity file. This can either be a floating pragma or a specific compiler version.

Remediation # 2:

It is recommended to use a fixed pragma version, as future compiler versions may handle certain language constructions in a way the developer did not foresee.
Using a floating pragma may introduce several vulnerabilities if compiled with an older version.
The developers should always use the exact Solidity compiler version when designing their contracts as it may break the changes in the future.
Instead of `^0.8.0` use `pragma solidity 0.8.7`, which is a stable and recommended version right now.

Issue # 3:

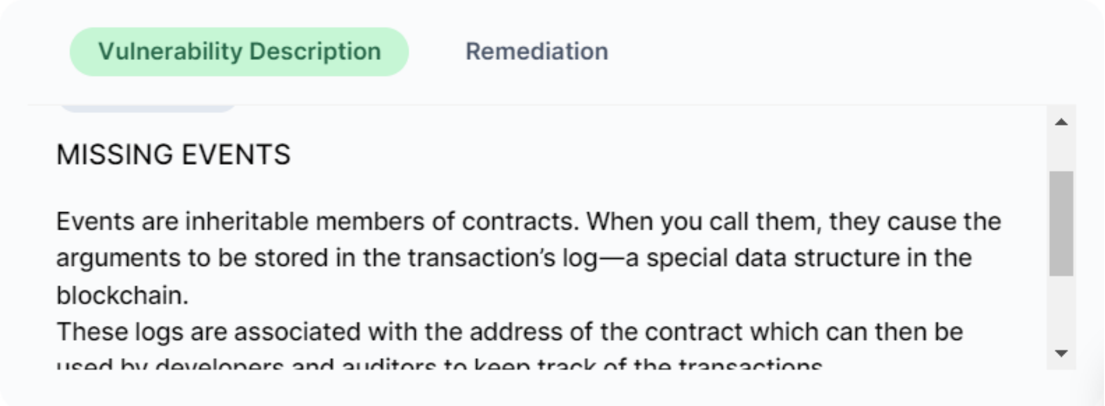
MISSING EVENTS:
Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain.
These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.
The contract `Address` was found to be missing these events on the function `sendValue` which would make it difficult or impossible to track these transactions off-chain.



```
     * https://diligence.consensys.net/posts/2019/09/stop-u
     *
     * IMPORTANT: because control is transferred to `recipi
     * taken to not create reentrancy vulnerabilities. Cons
     * {ReentrancyGuard} or the
    * https://solidity.readthedocs.io/en/v0.5.11/security-
     */
    function sendValue(address payable recipient, uint256 a
        require(address(this).balance >= amount, "Address:

        (bool success, ) = recipient.call{value: amount}(""
        require(success, "Address: unable to send value, re
    }

    /**
```



MISSING EVENTS

Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain.
These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions

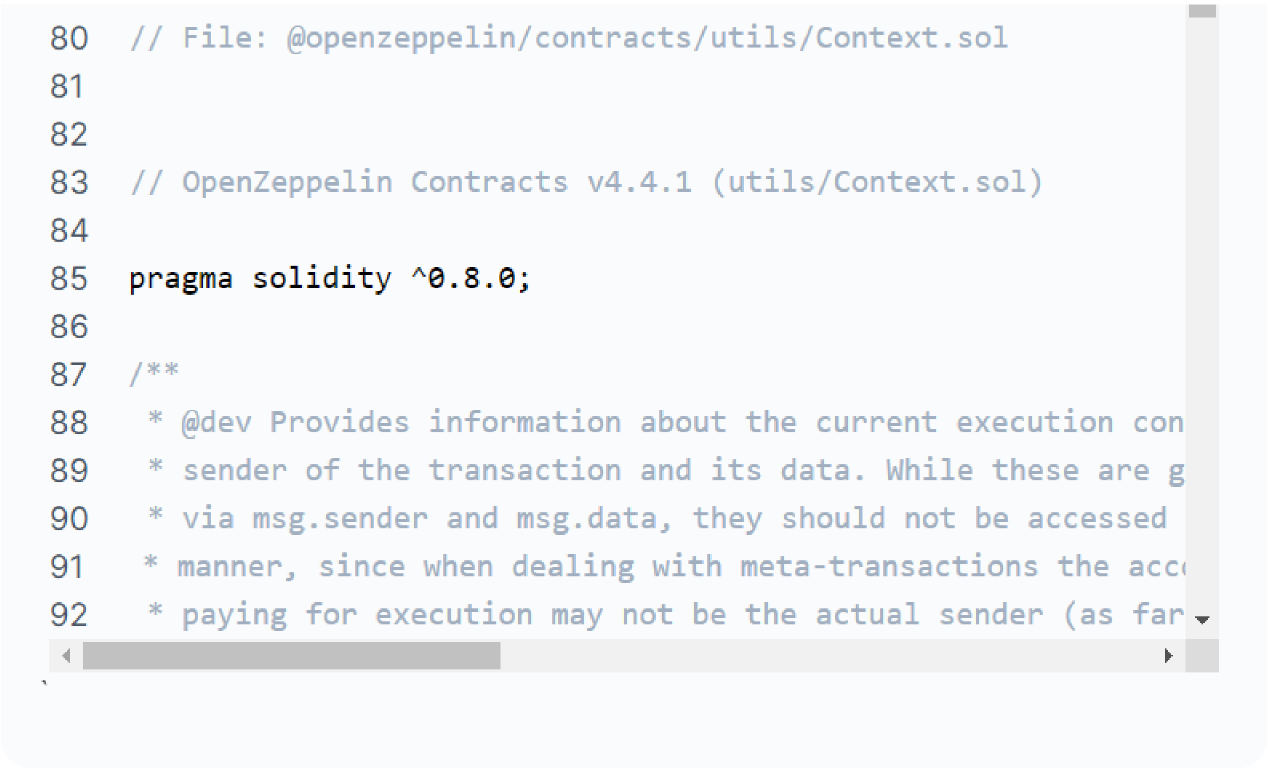
Remediation # 3:

Consider emitting events for the functions mentioned above. It is also recommended to have the addresses indexed.

Issue # 4:

OUTDATED COMPILER VERSION:

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.
The following outdated versions were detected:
`contract.sol` - `^0.8.0`

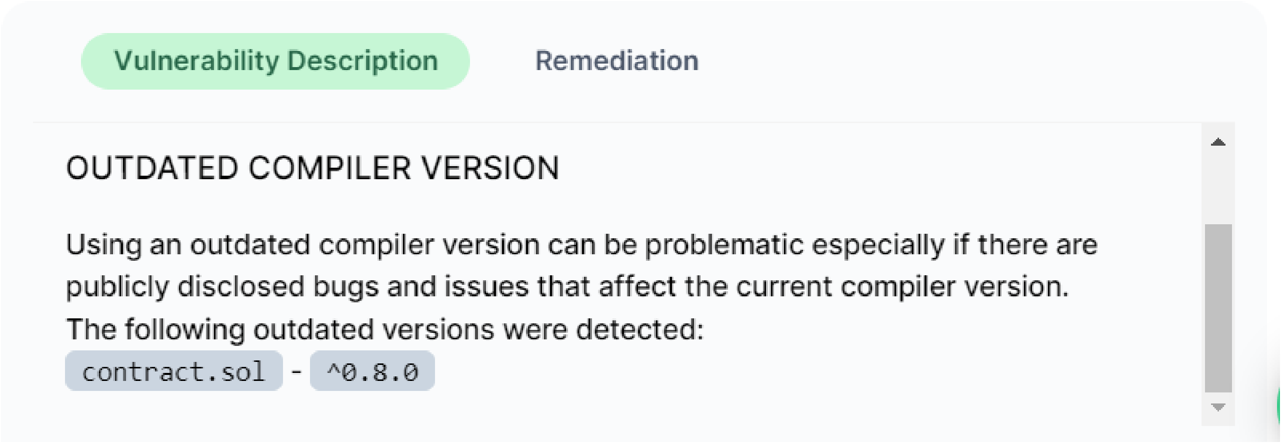


```
// File: @openzeppelin/contracts/utils/Context.sol


// OpenZeppelin Contracts v4.4.1 (utils/Context.sol)

pragma solidity ^0.8.0;

/**
 * @dev Provides information about the current execution con
 * sender of the transaction and its data. While these are g
 * via msg.sender and msg.data, they should not be accessed
 * manner, since when dealing with meta-transactions the acc
 * paying for execution may not be the actual sender (as far
```



OUTDATED COMPILER VERSION

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version. The following outdated versions were detected:
`contract.sol` - `^0.8.0`

Remediation # 4:

It is recommended to use a recent version of the Solidity compiler that should not be the most recent version, and it should not be an outdated version as well. Using very old versions of Solidity prevents the benefits of bug fixes and newer security checks. Consider using the solidity version `0.8.7`, which patches most solidity vulnerabilities.

Conclusion

Smart contracts contain High severity issues! Liquiditypair contract's security is not checked due to out of scope.

Liquidity locking details NOT provided by the team.

Itish note:

Please check the disclaimer above and note, the audit makes no statements or warranties on business model, investment attractiveness or code sustainability. The report is provided for the only contract mentioned in the report and does not include any other potential contracts deployed by Owner.