

TokenSale.sol Contract Findings and Analysis

Scope

The code under review can be found in my private repository, and is composed of 3 smart contracts written in the Solidity programming language and includes 409 lines of Solidity code.

Severity Criteria

Peter assesses the severity of disclosed vulnerabilities based on three primary risk categories: **high, medium, and low/non-critical.**

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

SUMMARY

- **1) High-Risk Finding**

[H-01] Initializer modifier not defined.

[H-02] Initialize function can be front-run.

[H-03] Rounding error.

[H-04] No check to make sure values are initialized.

[H-05] Erroneous conversion

- **2)Medium-Risk Finding**

NONE

- **3)Low/Non-Critical:**

[G-01] Functions guaranteed to revert when called by normal users can be marked payable

[G-02] Public Functions To External

[G-03] Store constant value with decimal factored

[G-04] The initialize function should be marked external

[G-05] The convertWeiToWholeTokens function should be marked internal

[G-06] Public functions not used in the contract should be made external

[N-01] Misleading documentation

[N-02] Function writing that does not comply with the Solidity Style Guide Context

[N-03] Use of modifier not necessary

[N-04] Licensing Information: specify a standard open-source license for the contract, such as MIT, Apache, or GPL.

[N-05] Oudated Solidity Version Specification

[N-06] Inconsistent Solidity Versions

1) High:

[H-01] Initializer modifier not defined.

Description:

The initializer modifier is not defined, enabling anybody to be able to call the initialize function.

Impact:

Anybody can call the initialize function at any time and become the owner of the contract. This can enable the person to be able to withdraw all tokens in the contract.

Proof of Concept:

<https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L65>

Recommended Mitigation:

The owner state variable should be set in the constructor and the initializer modifier should check that the person calling it is the owner. The onlyAdmin modifier can also be used.

[H-02] Initialize function can be front-run.

Description:

Since the initializer modifier was not defined, anybody can call the initialize function. If the transaction to call the initialize function is front-run, the front-runner will therefore become the owner giving him/her exclusive access to all the funds in the contract which can eventually lead to loss of funds for the project.

Impact:

All funds raised by the project can be withdrawn by the front-runner

Proof of Concept:

<https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L65>

Recommended Mitigation:

The owner state variable should be set in the constructor and the initializer modifier should check that the person calling it is the owner. The onlyAdmin modifier can also be used.

[H-03] Rounding error.

Description:

In the convertWeiToWholeTokens function, any value that is less than 1 ether will return zero since the EVM has no way of representing decimal numbers.

This will also render the getContractEtherBalanceWholeTokens, getContractTokenBalanceWholeTokens and calculateTokenAmountInWeiWholeTokens functions misleading as they rely on the convertWeiToWholeTokens function.

The calculateTokenAmountInWei and calculateTokenAmountInWeiWholeTokens functions can also have the rounding error. So it is essential to ensure that the values set as tokensPerEthMultiplier and tokensPerEthDivisor do not cause a rounding problem as those values set the price of the token.

Impact:

A rounding error will cause all places where the return value of the function is to be used to result in misleading calculations.

Proof of Concept:

- 1) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L340>
- 2) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L276>
- 3) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L253>
- 4) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L226>
- 5) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L223>
- 6) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L215>

Recommended Mitigation:

The calculation should be done in the function where the value is used or a rounding factor should be considered and used.

[H-04] No check to make sure values are initialized.

Description:

The calculateTokenAmountInWei, calculateTokenAmountInWeiWholeTokens and buyTokens functions do not have any check to make sure that the values of tokensPerEthMultiplier and tokensPerEthDivisor have been set.

Impact:

The values of the tokensPerEthMultiplier and tokensPerEthDivisor variables would return zero if they have not been set causing error in the calculation which would result in misleading values.

Proof of Concept:

- 1) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L215>
- 2) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L223>
- 3) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L179>

Recommended Mitigation:

Include a check to ensure that the tokensPerEthMultiplier and tokensPerEthDivisor variables have been set.

[H-05] Erroneous conversion**Description:**

The daiAmountInWei variable does not represent the value assigned to it. It instead holds the amount of Ether in Wei the user is trying to buy.

Impact:

This would lead to erroneous calculations as the amounts involved would be valued less. In a situation where a user wants to buy 100 Ether worth of the token, this value would be recorded as 100 Dai worth which is very much undervalued.

Proof of Concept:

<https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L183>

Recommended Mitigation:

To check if the amount of Ether the person is trying to purchase with is up to the maxBuyAmountPerSessionInDAI, the Dai value of Eth which can be gotten via an oracle like Chainlink is to be gotten in order to get the Dai value of the Eth the user intends to buy with. This value can then be compared with maxBuyAmountPerSessionInDAI.

3)Low/Non-Critical:

[G-01] Functions guaranteed to revert when called by normal users can be marked payable

If a function modifier or require such as onlyOwner/onlyX is used, the function will revert if a normal user tries to pay the function. Marking the function as payable will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided. The extra opcodes avoided are CALLVALUE(2), DUP1(3), ISZERO(3), PUSH2(3), JUMPI(10), PUSH1(3), DUP1(3), REVERT(0), JUMPDEST(1), POP(2) which costs an average of about 21 gas per call to the function, in addition to the extra deployment cost.

Proof Of Concept

#CODE BLOCKS

Recommended Mitigation Steps

Functions guaranteed to revert when called by normal users can be marked payable.

[G-02] Public Functions To External

The following functions could be set external to save gas and improve code quality. External call cost is less expensive than of public functions.

[G-03] Store constant value with decimal factored

Description:

The maxBuyAmountPerSessionInDAI state variable should be multiplied by the decimal factor when stored. This will reduce the number of computations being done when the buyTokens function is called thereby reducing the gas cost. The mul opcode takes 5 gas which can be called just once when the maxBuyAmountPerSessionInDAI variable is set.

Impact:

Increased gas cost for users when trying to buy the tokens.

Proof of Concept:

<https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L50>

Recommended Mitigation:

Factor in the decimal when storing maxBuyAmountPerSessionInDAI . This can be done in the setMaxBuyAmountPerSessionInDAIWholeTokens function.

[G-04]The initialize function should be marked external

Description:

The initialize function is visibility is set to public although it is not called by any function in the contract. It can be set to external in order to reduce gas consumption.

Impact:

Increased gas consumption as function is visibility is set to public.

Proof of Concept:

<https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L65>

Recommended Mitigation:

Change function visibility from public to external.

[G-05] The convertWeiToWholeTokens function should be marked internal

Description:

The convertWeiToWholeTokens function's visibility is set to public although it can be set to internal to save gas.

Impact:

Increased gas consumption as function is visibility is set to public.

Proof of Concept:

<https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L340>

Recommended Mitigation:

Change function visibility from public to internal.

[G-06] Public functions not used in thecontract should be made external

Description:

The `withdrawEther`, `withdrawAllEther`, `withdrawTokens` and `withdrawAllTokens` functions visibility is set to public although they can be set to external to save gas as the contract does not call it.

Impact:

Increased gas consumption as function visibility is set to public.

Proof of Concept:

- 1) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#LL325C5-L325C5>
- 2) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#LL310C14-L310C30>
- 3) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#LL297C14-L297C28>
- 4) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L286>

Recommended Mitigation:

Change function visibility from public to external.

Observations

- 1) The .call syntax was dropped for the .transfer syntax here:
<https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L333>
But the reason is unknown.

Proof Of Concept

#CODE BLOCKS

[N-01] Misleading documentation

Description:

The documentation for the isValidWalletAddress function is misleading. It talks about voting a wallet out when no wallet is voted out.

Another instance of this bug can be seen in the withdrawEther function. The amount passed in is the amount of Ether the admin wants to withdraw. Using ETH/DAI in this sense may be deceptive.

Impact:

May slow down development time when a developer is trying to figure out what the initial contract design was meant to achieve.

Proof of Concept:

- 1) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L349>

2)<https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#LL323C5-L323C5>

Recommended Mitigation:

Ensure proper documentation is followed.

[N-02] Function writing that does not comply with the Solidity Style Guide Context

All Contracts

Description

Order of Functions; ordering helps readers identify which functions they can call and to find the constructor and fallback definitions easier. But there are contracts in the project that do not comply with this.

<https://docs.soliditylang.org/en/v0.8.17/style-guide.html>

Functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public
- internal
- private
- within a grouping, place the view and pure functions last

[N-03] Use of modifier not necessary

Description:

The `getContractEtherBalance`, `getContractEtherBalanceWholeTokens`, `getContractTokenBalance`, `getContractTokenBalanceWholeTokens` and `getTokenBalanceAnyWallet` functions have the `onlyAdmin` modifier which is not necessary as these functions are view functions and the balance of the contract is a public value that is available on block explorers.

Impact:

Removing the modifier will reduce the cost of calling the functions .

Proof of Concept:

1) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#LL264C14-L264C37>

- 2) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#LL271C14-L271C48>
- 3) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L240>
- 4) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L248>
- 5) <https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L232>

Recommended Mitigation:

Remove the onlyAdmin modifier.

[N-04] Licensing Information**Description:**

The SPDX-License-Identifier in the contract header is set to "Galt Project Society Construction and Terraforming Company," which is not a standard open-source license identifier.

Impact:

Using a non-standard or incorrect SPDX-License-Identifier can lead to confusion regarding the actual licensing terms and conditions of the contract. It may cause difficulties when reusing or integrating the contract into other projects.

Proof of Concept:

SPDX-License-Identifier: Galt Project Society Construction and Terraforming Company

Recommended Mitigation:

Update the SPDX-License-Identifier to a standard open-source license identifier, such as MIT, Apache, or GPL, based on the intended licensing terms of the contract.

[N-05] Outdated Solidity Version Specification**Description:**

The contract specifies a Solidity version range from $\geq 0.5.0$ to $< 0.9.0$, which includes outdated versions and assumes the existence of version 0.9.0, which is not available at the time of writing.

Impact:

Using an outdated Solidity version and assuming the existence of an unreleased version can lead to compatibility issues, as well as missing out on the latest language features, optimizations, and security enhancements provided in newer versions. It may also result in unexpected behavior or vulnerabilities due to potential changes in the Solidity compiler and runtime environment.

Proof of Concept:

The contract specifies the version range as "`>=0.5.0 <0.9.0`" in the pragma statement.

[https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/](https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L11)

[97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L11](https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L11)

Recommended Mitigation:

Determine the minimum required Solidity version based on the contract's dependencies and features used. Consider using the latest stable version or a specific version that has been thoroughly tested and widely adopted by the community.

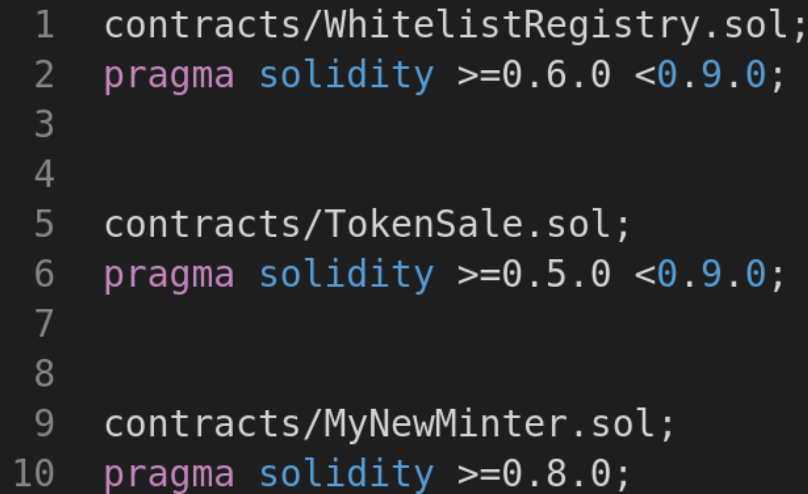
Update the Solidity version specification in the pragma statement to reflect the determined minimum required version. For example, if the contract requires version 0.8.0, the pragma statement should be: `pragma solidity ^0.8.0;`

Regularly monitor the Solidity language and ecosystem updates to stay informed about new features, bug fixes, and security improvements. Consider updating the contract to leverage these advancements when appropriate, while ensuring thorough testing and validation.

It is crucial to ensure the solidity version used is appropriate for the contract's requirements, taking into account compatibility, security, and the availability of stable releases.

[N-06] Inconsistent Solidity Versions**Description:**

The project contains inconsistencies in the Solidity compiler versions used throughout its codebase. Different parts of the contract are written using different Solidity versions or version ranges. The following contract mix versions



```
1  contracts/WhitelistRegistry.sol;  
2  pragma solidity >=0.6.0 <0.9.0;  
3  
4  
5  contracts/TokenSale.sol;  
6  pragma solidity >=0.5.0 <0.9.0;  
7  
8  
9  contracts/MyNewMinter.sol;  
10 pragma solidity >=0.8.0;
```

Impact:

Inconsistent Solidity versions can lead to compilation errors and unexpected behavior during contract deployment and execution. It can also make the contract code harder to read, maintain, and audit. Different versions may have different language features, syntax, and compiler optimizations, which can affect the contract's functionality, performance, and security.

Proof of Concept:

`pragma solidity >=0.5.0<0.9.0` is specified in the `pragma` statement at the beginning of `tcontracts/TokenSale.sol` file.

However, some other contracts files within the same project are written using `pragma solidity >=0.5.0 <0.9.0` or `pragma solidity >=0.8.0`;

Inconsistencies in syntax or language features specific to different versions may be observed in different parts of the contract code.

Recommended Mitigation:

Conduct a thorough review of the contract codebase to identify inconsistencies in Solidity versions. Determine the minimum required Solidity version for the contract based on its dependencies, external libraries, and desired language features.

Update all parts of the contract code to use a consistent Solidity version or version range that aligns with the determined minimum required version.

Ensure that the pragma statement at the beginning of the contract specifies the determined minimum required Solidity version or version range. For example, `pragma solidity ^0.8.15;`

Carefully test and validate the contract after making the necessary changes to ensure that it compiles correctly and functions as intended with the consistent Solidity version.

Establish a development process and coding conventions that promote consistency in Solidity versions across the project. This may include regular code reviews and automated checks for version inconsistencies.

Stay updated with the latest Solidity releases and best practices to leverage new features, performance improvements, and security updates while maintaining version consistency in the contract codebase.

WhitelistRegistry.sol Contract Findings and Analysis

Scope

The code under review can be found in my private repository, and is composed of 3 smart contracts written in the Solidity programming language and includes 64 lines of Solidity code.

Severity Criteria

Peter assesses the severity of disclosed vulnerabilities based on three primary risk categories: **high, medium, and low/non-critical.**

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

SUMMARY

- **1) High-Risk Finding**

[H-01] High-risk vulnerability in addCustomerToWhiteList and removeCustomerFromWhiteList functions: owner can pass address zero

- **2)Medium-Risk Finding**

NONE

- **3)Low/Non-Critical:**

[G-01] Functions guaranteed to revert when called by normal users can be marked payable

[G-02] Public Functions To External

[N-01] NatSpec comments should be increased in contracts

[N-02] Function writing that does not comply with the Solidity Style Guide **Context**

[N-03] Include return parameters in NatSpec comments

[N-04] Licensing Information: specify a standard open-source license for the contract, such as MIT, Apache, or GPL.

[N-05] Oudated Solidity Version Specification

[N-06] Inconsistent Solidity Versions

[N-07] Commented adjustment in getCustomersWhiteList function

1) High-Risk Findings

[H-01] High-risk vulnerability in addCustomerToWhiteList and removeCustomerFromWhiteList functions: owner can pass address zero

The `onlyAdminOrManager` can pass address zero (`0x00`) to the `addCustomerToWhiteList()` function, which can lead to unexpected behavior or even contract failure.

The contract does not validate the input parameters passed to the functions `addCustomerToWhiteList` and `removeCustomerFromWhiteList`. Any address can be added or removed from the whitelist without proper validation.

Impact

This vulnerability can allow the owner to add the zero address multiple times. Additionally, since the zero address is commonly used as a placeholder, it could potentially lead to other security issues down the line, such as unexpected contract behavior or loss of funds.

Proof of Concept:

```
1 function addCustomerToWhiteList(address _customer) external onlyAdminOrManager {
2     customersWhiteList.add(_customer);
3     emit AddWhitelistedCustomer(_customer, msg.sender);
4 }
5
6 function removeCustomerFromWhiteList(address _customer) external onlyAdminOrManager {
7     customersWhiteList.remove(_customer);
8     emit RemoveWhitelistedCustomer(_customer, msg.sender);
9 }
```

Recommended Mitigation

To fix this vulnerability, the function should include a validation check to ensure that `addCustomerToWhiteList()` is not the zero address. One way to do this is to add the following check at the beginning of the function:



```
1 function addCustomerToWhiteList(address _customer) external onlyAdminOrManager {
2     require(_customer != address(0), "WhitelistRegistry: Invalid customer address");
3     customersWhiteList.add(_customer);
4     emit AddWhitelistedCustomer(_customer, msg.sender);
5 }
6
7 function removeCustomerFromWhiteList(address _customer) external onlyAdminOrManager {
8     require(_customer != address(0), "WhitelistRegistry: Invalid customer address");
9     customersWhiteList.remove(_customer);
10    emit RemoveWhitelistedCustomer(_customer, msg.sender);
11 }
```

3)Low/Non-Critical:

[N-01] NatSpec comments should be increased in contracts

Context

All Contracts

Description:

It is recommended that Solidity contracts are fully annotated using NatSpec for all public interfaces (everything in the ABI). It is clearly stated in the Solidity official documentation.

In complex projects such as Defi, the interpretation of all functions and their arguments and returns is important for code readability and auditability.

<https://docs.soliditylang.org/en/v0.8.15/natspec-format.html>

Recommendation

NatSpec comments should be increased in contracts.

[N-02] Function writing that does not comply with the Solidity Style Guide Context

All Contracts

Description

Order of Functions; ordering helps readers identify which functions they can call and to find the constructor and fallback definitions easier. But there are contracts in the project that do not comply with this.

<https://docs.soliditylang.org/en/v0.8.17/style-guide.html>

Functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public
- internal
- private
- within a grouping, place the view and pure functions last

[N-03] Include return parameters in NatSpec comments

Context

All Contracts

Description

It is recommended that Solidity contracts are fully annotated using NatSpec for all public interfaces (everything in the ABI). It is clearly stated in the Solidity official documentation. In complex projects such as Defi, the interpretation of all functions and their arguments and returns is important for code readability and auditability.

<https://docs.soliditylang.org/en/v0.8.15/natspec-format.html>

Recommendation

[Include return parameters in NatSpec comments.](#)

[N-04] Licensing Information

Description:

The SPDX-License-Identifier in the contract header is set to "Galt Project Society Construction and Terraforming Company," which is not a standard open-source license identifier.

Impact:

Using a non-standard or incorrect SPDX-License-Identifier can lead to confusion regarding the actual licensing terms and conditions of the contract. It may cause difficulties when reusing or integrating the contract into other projects.

Proof of Concept:

SPDX-License-Identifier: Galt Project Society Construction and Terraforming Company

Recommended Mitigation:

Update the SPDX-License-Identifier to a standard open-source license identifier, such as MIT, Apache, or GPL, based on the intended licensing terms of the contract.

[N-05] Outdated Solidity Version Specification

Description:

The contract specifies a Solidity version range from $\geq 0.5.0$ to $< 0.9.0$, which includes outdated versions and assumes the existence of version 0.9.0, which is not available at the time of writing.

Impact:

Using an outdated Solidity version and assuming the existence of an unreleased version can lead to compatibility issues, as well as missing out on the latest language features, optimizations, and security enhancements provided in newer versions. It may also result in unexpected behavior or vulnerabilities due to potential changes in the Solidity compiler and runtime environment.

Proof of Concept:

The contract specifies the version range as " $\geq 0.5.0 < 0.9.0$ " in the pragma statement.

[https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/](https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L11)

[97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L11](https://github.com/scarfacedotcom/WhiteList-and-Token-Sale/blob/97e65bbb69768b6785772dc698957ab5089b1f31/contracts/TokenSale.sol#L11)

Recommended Mitigation:

Determine the minimum required Solidity version based on the contract's dependencies and features used. Consider using the latest stable version or a specific version that has been thoroughly tested and widely adopted by the community.

Update the Solidity version specification in the pragma statement to reflect the determined minimum required version. For example, if the contract requires version 0.8.0, the pragma statement should be: `pragma solidity ^0.8.0;`

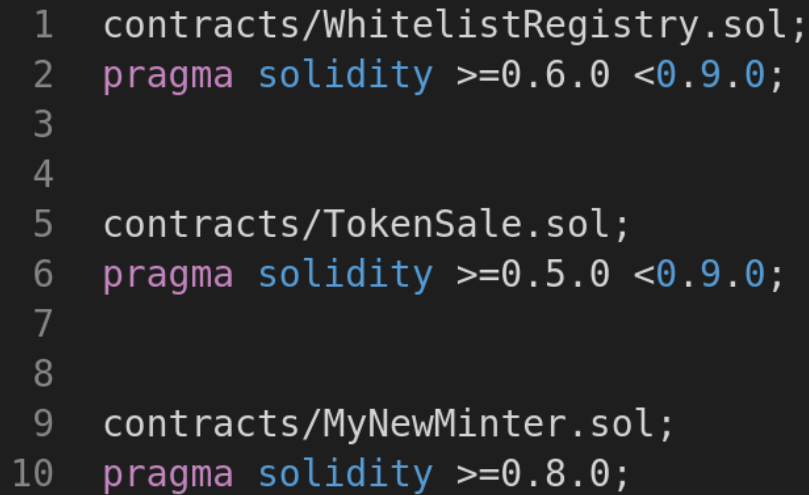
Regularly monitor the Solidity language and ecosystem updates to stay informed about new features, bug fixes, and security improvements. Consider updating the contract to leverage these advancements when appropriate, while ensuring thorough testing and validation.

It is crucial to ensure the solidity version used is appropriate for the contract's requirements, taking into account compatibility, security, and the availability of stable releases.

[N-06] Inconsistent Solidity Versions

Description:

The project contains inconsistencies in the Solidity compiler versions used throughout its codebase. Different parts of the contract are written using different Solidity versions or version ranges. The following contract mix versions



```
1  contracts/WhitelistRegistry.sol;  
2  pragma solidity >=0.6.0 <0.9.0;  
3  
4  
5  contracts/TokenSale.sol;  
6  pragma solidity >=0.5.0 <0.9.0;  
7  
8  
9  contracts/MyNewMinter.sol;  
10 pragma solidity >=0.8.0;
```

Impact:

Inconsistent Solidity versions can lead to compilation errors and unexpected behavior during contract deployment and execution. It can also make the contract code harder to read, maintain, and audit. Different versions may have different language features, syntax, and compiler optimizations, which can affect the contract's functionality, performance, and security.

Proof of Concept:

`pragma solidity >=0.6.0 <0.9.0` is specified in the `pragma` statement at the beginning of `tcontracts/WhitelistRegistry.sol` file.

However, some other contracts files within the same project are written using `pragma solidity >=0.5.0 <0.9.0` or `pragma solidity >=0.8.0`;

Inconsistencies in syntax or language features specific to different versions may be observed in different parts of the contract code.

Recommended Mitigation:

Conduct a thorough review of the contract codebase to identify inconsistencies in Solidity versions. Determine the minimum required Solidity version for the contract based on its dependencies, external libraries, and desired language features.

Update all parts of the contract code to use a consistent Solidity version or version range that aligns with the determined minimum required version.

Ensure that the pragma statement at the beginning of the contract specifies the determined minimum required Solidity version or version range. For example, `pragma solidity ^0.8.15;`

Carefully test and validate the contract after making the necessary changes to ensure that it compiles correctly and functions as intended with the consistent Solidity version.

Establish a development process and coding conventions that promote consistency in Solidity versions across the project. This may include regular code reviews and automated checks for version inconsistencies.

Stay updated with the latest Solidity releases and best practices to leverage new features, performance improvements, and security updates while maintaining version consistency in the contract codebase.

[N-07] Commented adjustment in getCustomersWhiteList function**Description:**

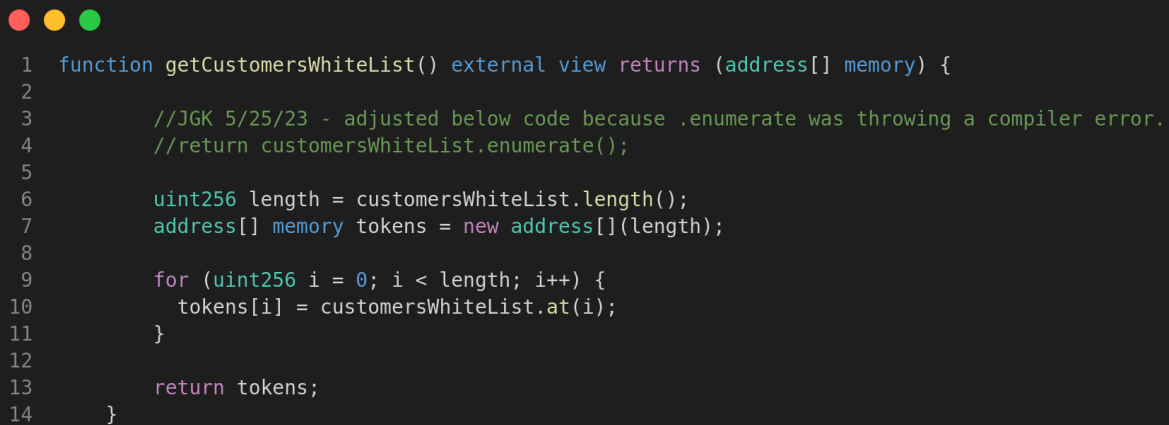
The `getCustomersWhiteList` function has a commented code mentioning an adjustment, but the actual adjustment is missing. It's unclear whether the adjustment was made or not.

Impact:

The presence of incomplete or commented-out code can cause confusion to developers and auditors trying to understand the contract. It may lead to misconceptions about the intended behavior of the function.

Proof of Concept:

Affected code:



```
1 function getCustomersWhiteList() external view returns (address[] memory) {
2
3     //JGK 5/25/23 - adjusted below code because .enumerate was throwing a compiler error.
4     //return customersWhiteList.enumerate();
5
6     uint256 length = customersWhiteList.length();
7     address[] memory tokens = new address[](length);
8
9     for (uint256 i = 0; i < length; i++) {
10         tokens[i] = customersWhiteList.at(i);
11     }
12
13     return tokens;
14 }
```

Recommended Mitigation:

Remove the commented adjustment or update the code with the actual adjustment made. Ensure that the function behaves as intended and accurately represents its purpose.

Missing initializer modifier for initialize function

Description:


The initialize function is missing the modifier `initializer`, which is required for initialization functions in the OpenZeppelin initializer pattern.

Impact:

The missing initializer modifier can lead to potential reentrancy and security vulnerabilities during contract deployment or initialization. It may also prevent the contract from functioning as intended.

Proof of Concept:


Affected code:



```
1 function initialize(address _owner) public override {  
2     Ownable.initialize(_owner);  
3 }
```

Recommended Mitigation:

Add the initializer modifier to the initialize function as follows:



```
1 function initialize(address _owner) public override initializer {  
2     Ownable.initialize(_owner);  
3 }
```

Ensure that the contract follows the proper initialization pattern to safeguard against reentrancy and potential security issues.

MyNewMinter.sol Contract Findings and Analysis

Scope

The code under review can be found in my private repository, and is composed of 3 smart contracts written in the Solidity programming language and includes 37 lines of Solidity code.

Severity Criteria

Peter assesses the severity of disclosed vulnerabilities based on three primary risk categories: **high, medium, and low/non-critical.**

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

SUMMARY

- **1) High-Risk Finding**

[H-01] Lack of Access control on the Mint Function

- **2) Medium-Risk Finding**

[M-01] Lack of proper Event Logging

- **3)Low/Non-Critical:**

[G-01] Use solidity version 0.8.19 to gain some gas boost

[G-02] Public Functions To External

[N-01] NatSpec comments should be increased in contracts

[N-02] Function writing that does not comply with the Solidity Style Guide **Context**

[N-03] Include return parameters in NatSpec comments

[N-04] Licensing Information: specify a standard open-source license for the contract, such as MIT, Apache, or GPL.

[N-06] Inconsistent Solidity Versions

[N-07] Commented adjustment in getCustomersWhiteList function

- **1) High-Risk Finding**

[H-01] Lack of Access control on the Mint Function

Description:

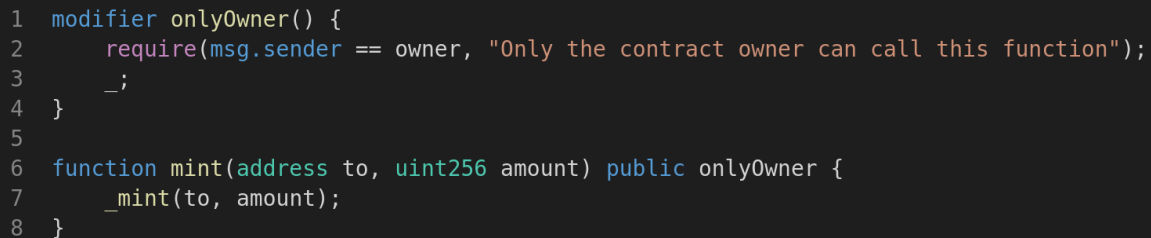
The mint function is only restricted by the onlyOwner modifier, which allows only the contract owner to call it. However, there is no mechanism in place to manage or change the contract owner. If the initial contract owner is compromised or if there is no functionality to transfer ownership, it can lead to a permanent loss of control over the contract.

Impact:

Without proper access control, unauthorized individuals can potentially gain control over the minting functionality and manipulate the token supply. This can result in loss of funds, inflationary issues, or other unintended consequences.

Proof of Concept:

Affected code:



```
1  modifier onlyOwner() {
2      require(msg.sender == owner, "Only the contract owner can call this function");
3      _;
4  }
5
6  function mint(address to, uint256 amount) public onlyOwner {
7      _mint(to, amount);
8  }
```

Recommended Mitigation:

Implement a mechanism to manage and transfer ownership. Consider using an access control framework, such as OpenZeppelin's Ownable contract, to manage access rights and ensure secure ownership transfer.

You can implement an ownership transfer function that can only be called by the current contract owner. Here's an example:



```
1 function transferOwnership(address newOwner) public onlyOwner {  
2     require(newOwner != address(0), "Invalid new owner");  
3     owner = newOwner;  
4 }  
5
```

This function allows the current owner to transfer ownership to a new address. It is important to ensure that the new owner is a valid address and not set to the zero address.

2) Medium-Risk Finding

[M-01] Lack of proper Event Logging

Description:

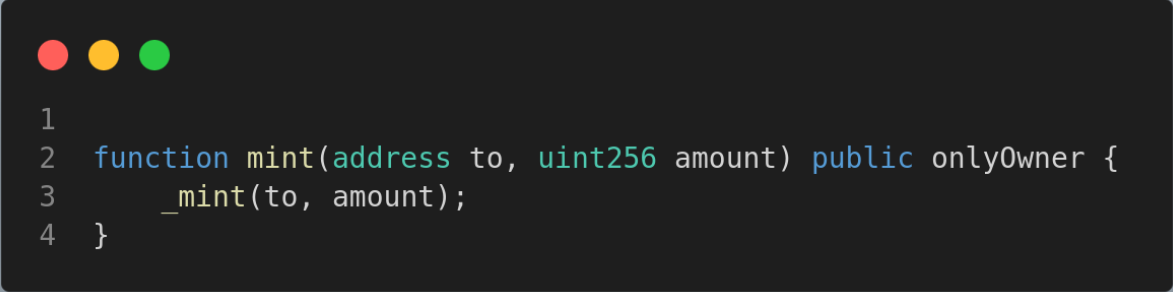
The contract does not emit any events when minting tokens. Event logging is essential for transparency and traceability purposes, and it provides an auditable trail of token minting activities.

Impact:

The absence of event logging can hinder transparency and make it difficult to track and verify token minting activities. It can also make it challenging to monitor and audit the contract's behavior, potentially hiding malicious or unintended token minting actions.

Proof of Concept:

No events are emitted for token minting.




```
1
2 function mint(address to, uint256 amount) public onlyOwner {
3     _mint(to, amount);
4 }
```

Recommended Mitigation:

Emit events, such as a Mint event, when minting tokens. This provides transparency and enables easier tracking of token minting activities. Emit relevant events with relevant information, such as the recipient's address and the amount of tokens minted.

You can emit a custom event, such as a Mint event, when minting tokens. Here's an example:



```
1 event Mint(address indexed to, uint256 amount);
2
3 function mint(address to, uint256 amount) public onlyOwner {
4     _mint(to, amount);
5     emit Mint(to, amount);
6 }
```

In this example, the Mint event is emitted after successfully minting tokens. The event includes the recipient's address (to) and the amount of tokens minted (amount). Emitting events provides transparency and facilitates monitoring and auditing of token minting activities.

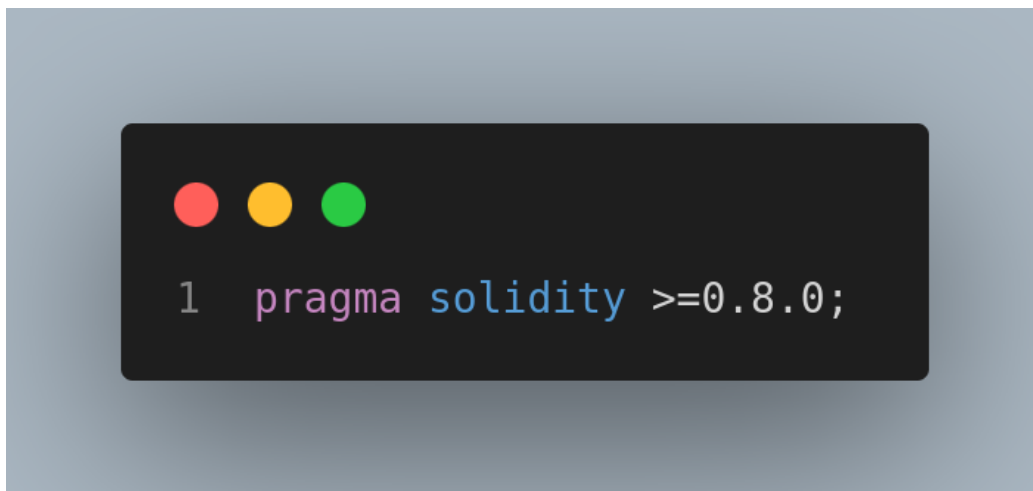
3)Low/Non-Critical:

[G-01] Use the latest solidity version to gain some gas boost

Upgrade to the latest solidity version to get additional gas savings.

See latest release for reference: <https://blog.soliditylang.org/2023/02/22/solidity-0.8.19-release-announcement/>

Proof of Concept



Recommended Mitigation:

Determine the minimum required Solidity version based on the contract's dependencies and features used. Consider using the latest stable version or a specific version that has been thoroughly tested and widely adopted by the community.

Update the Solidity version specification in the pragma statement to reflect the determined minimum required version. For example, if the contract requires version 0.8.0, the pragma statement should be: `pragma solidity ^0.8.0;`

Regularly monitor the Solidity language and ecosystem updates to stay informed about new features, bug fixes, and security improvements. Consider updating the contract to leverage these advancements when appropriate, while ensuring thorough testing and validation.

It is crucial to ensure the solidity version used is appropriate for the contract's requirements, taking into account compatibility, security, and the availability of stable releases.

[N-01] NatSpec comments should be increased in contracts

Context

All Contracts

Description:

It is recommended that Solidity contracts are fully annotated using NatSpec for all public interfaces (everything in the ABI). It is clearly stated in the Solidity official documentation.

In complex projects such as Defi, the interpretation of all functions and their arguments and returns is important for code readability and auditability.

<https://docs.soliditylang.org/en/v0.8.15/natspec-format.html>

Recommendation

NatSpec comments should be increased in contracts.

[N-02] Function writing that does not comply with the Solidity Style Guide Context

All Contracts

Description

Order of Functions; ordering helps readers identify which functions they can call and to find the constructor and fallback definitions easier. But there are contracts in the project that do not comply with this.

<https://docs.soliditylang.org/en/v0.8.17/style-guide.html>

Functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public
- internal
- private
- within a grouping, place the view and pure functions last

[N-03] Include return parameters in NatSpec comments

Context

All Contracts

Description

It is recommended that Solidity contracts are fully annotated using NatSpec for all public interfaces (everything in the ABI). It is clearly stated in the Solidity official documentation. In complex projects such as Defi, the interpretation of all functions and their arguments and returns is important for code readability and auditability.

<https://docs.soliditylang.org/en/v0.8.15/natspec-format.html>

Recommendation

[Include return parameters in NatSpec comments.](#)

[N-04] Licensing Information

Description:

The SPDX-License-Identifier in the contract header is set to "KuhnSoft LLC" which is not a standard open-source license identifier.

Impact:

Using a non-standard or incorrect SPDX-License-Identifier can lead to confusion regarding the actual licensing terms and conditions of the contract. It may cause difficulties when reusing or integrating the contract into other projects.

Proof of Concept:

SPDX-License-Identifier: KuhnSoft LLC



```
1 // SPDX-License-Identifier: KuhnSoft LLC
```

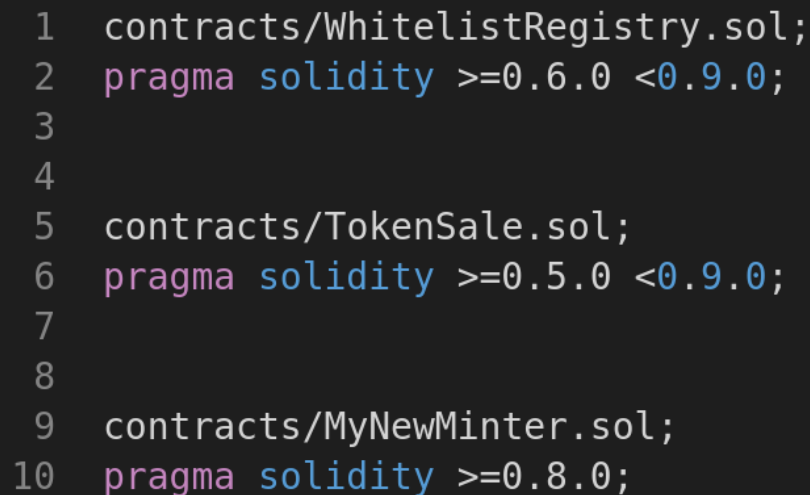
Recommended Mitigation:

Update the SPDX-License-Identifier to a standard open-source license identifier, such as MIT, Apache, or GPL, based on the intended licensing terms of the contract.

[N-06] Inconsistent Solidity Versions

Description:

The project contains inconsistencies in the Solidity compiler versions used throughout its codebase. Different parts of the contract are written using different Solidity versions or version ranges. The following contract mix versions



```
1  contracts/WhitelistRegistry.sol;
2  pragma solidity >=0.6.0 <0.9.0;
3
4
5  contracts/TokenSale.sol;
6  pragma solidity >=0.5.0 <0.9.0;
7
8
9  contracts/MyNewMinter.sol;
10 pragma solidity >=0.8.0;
```

Impact:

Inconsistent Solidity versions can lead to compilation errors and unexpected behavior during contract deployment and execution. It can also make the contract code harder to read, maintain, and audit. Different versions may have different language features, syntax, and compiler optimizations, which can affect the contract's functionality, performance, and security.

Proof of Concept:

`pragma solidity >=0.6.0 <0.9.0` is specified in the `pragma` statement at the beginning of `tcontracts/WhitelistRegistry.sol` file.

However, some other contracts files within the same project are written using `pragma solidity >=0.5.0 <0.9.0` or `pragma solidity >=0.8.0`;

Inconsistencies in syntax or language features specific to different versions may be observed in different parts of the contract code.

Recommended Mitigation:

Conduct a thorough review of the contract codebase to identify inconsistencies in Solidity versions. Determine the minimum required Solidity version for the contract based on its dependencies, external libraries, and desired language features.

Update all parts of the contract code to use a consistent Solidity version or version range that aligns with the determined minimum required version.

Ensure that the `pragma` statement at the beginning of the contract specifies the determined minimum required Solidity version or version range. For example, `pragma solidity ^0.8.15`;

Carefully test and validate the contract after making the necessary changes to ensure that it compiles correctly and functions as intended with the consistent Solidity version.

Establish a development process and coding conventions that promote consistency in Solidity versions across the project. This may include regular code reviews and automated checks for version inconsistencies.

Stay updated with the latest Solidity releases and best practices to leverage new features, performance improvements, and security updates while maintaining version consistency in the contract codebase.