Bug ID #1

Missing Validation in _transfer Function

Vulnerability Type
Missing validation

Severity
Low

Description
The provided smart contract contains a vulnerability where the `_transfer` function does not properly validate the transfer of tokens to the contract's address (`address(this)`). The `transferFrom` function disallows transfers to `address(this)`, which is a good security practice to prevent unauthorized token transfers to the contract. However, the `_transfer` function, which is called internally, lacks this validation. This oversight can lead to any external user sending tokens to the contract.

Affected Code

- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L887

Impacts
This vulnerability could have various impacts, such as unauthorized funds being deposited into the contract, and disrupting the contract's intended functionality. Also funds can be stuck in the contract.

Remediation
To mitigate this issue, it is recommended to add validation within the `_transfer` function to ensure that it does not accept transfers to `address(this)`. This can be achieved by adding a check that prevents token transfers to the contract's address.

Retest:

Bug ID #2

Use Ownable2Step

Vulnerability Type

Missing Best Practices

Severity

Low

Description

The "Ownable2Step" pattern is an improvement over the traditional "Ownable" pattern, designed to enhance the security of ownership transfer functionality in a smart contract. Unlike the original "Ownable" pattern, where ownership can be transferred directly to a specified address, the "Ownable2Step" pattern introduces an additional step in the ownership transfer process. Ownership transfer only completes when the proposed new owner explicitly accepts the ownership, mitigating the risk of accidental or unintended ownership transfers to mistyped addresses.

Affected Code

- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L716

Impacts

Without the "Ownable2Step" pattern, the contract owner might inadvertently transfer ownership to an unintended or mistyped address, potentially leading to a loss of control over the contract. By adopting the "Ownable2Step" pattern, the smart contract becomes more resilient against external attacks aimed at seizing ownership or manipulating the contract's behavior.

Remediation

It is recommended to use either Ownable2Step or Ownable2StepUpgradeable depending on the smart contract.

Retest:

Bug ID #3

Floating and Outdated Pragma

Vulnerability Type

Floating Pragma (SWC-103)

Severity

Low

Description

Locking the pragma helps ensure that the contracts do not accidentally get deployed using an older version of the Solidity compiler affected by vulnerabilities.
The contract allowed floating or unlocked pragma to be used, i.e.,^0.8.13. This allows the contracts to be compiled with all the solidity compiler versions above the limit specified. The following contracts were found to be affected -

Affected Code

- All Solidity Files

Impacts

If the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions.
Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic.
The likelihood of exploitation is really low therefore this is only informational.

Remediation

Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.21 pragma version
Reference: https://swcregistry.io/docs/SWC-103

Retest:

Bug ID #4

Require with Empty Message

Vulnerability Type
Code optimization

Severity
Informational

Description
During analysis; multiple require statements were detected with empty messages. The statement takes two parameters, and the message part is optional. This is shown to the user when and if the require statement evaluates to false. This message gives more information about the conditional and why it gave a false response.

Affected Code

- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L842

Impacts
Having a short descriptive message in the require statement gives users and developers more details as to why the conditional statement failed and helps in debugging the transactions.

Remediation
It is recommended to add a descriptive message, no longer than 32 bytes, inside the required statement to give more detail to the user about why the condition failed.

Retest:

Bug ID #5

Boolean Equality

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The contract was found to be equating variables with a boolean constant inside a “require()” statement which is not recommended and is unnecessary. Boolean constants can be used directly in conditionals.

Affected Code

- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L894
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L895

Impacts
Equating the values to boolean constants in conditions cost gas and can be used directly.

Remediation
It is recommended to use boolean constants directly. It is not required to equate them to true or false.

Retest:

Bug ID #6

Use of SafeMath

Vulnerability Type
Gas Optimization

Severity
Gas

Description
SafeMath library is found to be used in the contract. This increases gas consumption more than traditional methods and validations if done manually.
Also, Solidity **0.8.0** and above includes checked arithmetic operations by default, rendering SafeMath unnecessary.

Affected Code

- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L717

Impacts:
This increases the gas usage of the contract.

Remediation
We do not recommend using the SafeMath library for all arithmetic operations. It is good practice to use explicit checks where it is really needed and to avoid extra checks where overflow/underflow is impossible.
The compiler above 0.8.0+ automatically checks for overflows and underflows.

Retest:

Bug ID #7

Unnecessary Checked Arithmetic In Loop

Vulnerability Type

Gas Optimization

Severity

Gas

Description

Upon reviewing the code, it has been identified that the contract uses checked arithmetic operations inside loops where increments occur. However, it's important to note that increments inside loops are unlikely to cause overflow since the transaction will run out of gas before the variable reaches its limits. As a result, using checked arithmetic for increments within loops may be unnecessary and can lead to additional gas consumption.

Affected Code

- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L950

Impacts

Unnecessary checked arithmetic operations can lead to higher gas consumption, as each arithmetic operation comes with its own gas cost. This can contribute to increased transaction fees and operational costs.

Remediation

Consider having the increment value inside the unchecked block to save some gas.

Retest:

Bug ID #8

Array Length Caching

Vulnerability Type

Gas Optimization

Severity

Gas

Description

During each iteration of the loop, reading the length of the array uses more gas than is necessary. In the most favorable scenario, in which the length is read from a memory variable, storing the array length in the stack can save about 3 gas per iteration. In the least favorable scenario, in which external calls are made during each iteration, the amount of gas wasted can be significant.

Affected Code

- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L950

Impacts

Reading the length of an array multiple times in a loop by calling .length costs more gas.

Remediation

Consider storing the array length of the variable before the loop and use the stored length instead of fetching it in each iteration.

Retest:

Bug ID #9

Gas Optimization for State Variables

Vulnerability Type
Gas Optimization

Severity
Gas

Description
In Solidity, the compound assignment operators '+=' and '-=' tend to consume more gas compared to the basic addition and subtraction operators ('+' and '-', respectively). As a result, when you use 'x += y', it typically incurs a higher gas cost than using 'x = x + y'.

Affected Code

- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L1013
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L1028

Impacts
By using basic operators or optimizing code, you can decrease the gas costs associated with smart contract transactions. This can make your application more cost-effective.

Remediation
Replace += and -= with the basic + and - operators whenever feasible. This can help reduce gas consumption, especially when working with large-scale operations.

Retest:

Bug ID #10

Unnecessary Default Value Initialization

Vulnerability Type
Gas Optimization

Severity
Gas

Description
In Solidity, data types are automatically assigned default values if they are not explicitly initialized. However, the initialization of default values can sometimes be unnecessary or inefficient, depending on the specific use case.

Affected Code

- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L732
  https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L733
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L734

Impacts
Initializing data types to their default values is not required and costs additional gas.

Remediation
It's not recommended to initialize the data types to their default values unless there's a use-case because it's unnecessary and costs around ~3 gas.

Retest:

Bug ID #11

Gas Optimization in Require/Revert Statements

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The **require()** statement takes an input string to show errors if the validation fails.
The strings inside these functions that are longer than **32 bytes** require at least one additional MSTORE, along with additional overhead for computing memory offset and other parameters. For this purpose, having strings lesser than 32 bytes saves a significant amount of gas.

Vulnerable Code

- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L217
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L365
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L880
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L892
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L893
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L894
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L895
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L923
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L949
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L962
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L975

- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L990
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L997
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L1007
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L1026
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L1038
- https://etherscan.io/token/0xd098a30ae6c4a202dad8155dc68e2494ebac871c#code#F1#L1039

Impacts
Having longer require/revert strings than 32 bytes costs a significant amount of gas.

Remediation
It is recommended to shorten the strings passed inside **require()** statements to fit under **32 bytes**. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Retest: