



SAFETIN **AUDIT**

**MultiFunctional Environmental Token -
MFETVault**

December 20th, 2022



TABLE OF CONTENTS

- I. SUMMARY
- II. OVERVIEW
- III. FINDINGS
 - A. [SEND-1](#) : Do not use msgSender() unless implementing EIP-2771
 - B. [PACK-1](#) : Struct packing could be improved for Stats
 - C. [VAR-1](#) : State variables iterated on but not used for functionality could be removed
 - D. [UINT-1](#) : MAX_UINT state variable is not used
 - E. [VAR-2](#) : State variables in DataSets could be packed into fewer slots
 - F. [METH-1](#) : Best practice sending BNB is to use call method
 - G. [STATE-1](#) : Flipping state for investmentStatus could be adjusted
 - H. [PRAC-1](#) : Best practice to follow check effects interactions within functions
 - I. [SEND-2](#) : Assignment of msgSender() to user variable uses unnecessary gas

J. **CHECK-1** : Overflow checks can be removed as
Solidity 0.8.0+ automatically checks

K. **SAFE-1** : SafeMath does not need to be imported for
projects using Solidity 0.8.0+

L. **CALC-1**: Update payouts could be negative from
calculation

IV. DISCLAIMER

AUDIT SUMMARY

This report was written for [MultiFunctional Environmental Token](#) in order to find flaws and vulnerabilities in the [MultiFunctional Environmental Token](#) project's source code, as well as any contract dependencies that weren't part of an officially recognized library.

A comprehensive examination has been performed, utilizing Static Analysis, Manual Review, and [MultiFunctional Environmental Token](#) Deployment techniques. The auditing process pays special attention to the following considerations:

- ❖ Testing the smart contracts against both common and uncommon attack vectors
- ❖ Assessing the codebase to ensure compliance with current best practices and industry standards
- ❖ Ensuring contract logic meets the specifications and intentions of the client
- ❖ Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- ❖ Through line-by-line manual review of the entire codebase by industry expert

AUDIT OVERVIEW

PROJECT SUMMARY

Project name	MultiFunctional Environmental Token
Description	MFET is an ecosystem that provides consultancy to companies in their environmental works as an environmentally friendly token supporting sustainable projects.
Platform	BNB Smart-Chain
Language	Solidity
Codebase	https://bscscan.com/address/0xC0288b4C332cB9a13Aa81AB2097a34685708420b#code

FINDINGS SUMMARY

Vulnerability	Total	Resolved
● Critical	0	0
● Major	0	0
● Medium	1	0
● Minor	0	0
● Informational	11	0

AUDIT FINDINGS

Code	Title	Severity
SEND-1	Do not use msgSender() unless implementing EIP-2771	● Informational
PACK-1	Struct packing could be improved for Stats	● Informational
VAR-1	State variables iterated on but not used for functionality could be removed	● Informational
UINT-1	MAX_UINT state variable is not used	● Informational
VAR-2	State variables in DataSets could be packed into fewer slots	● Informational
METH-1	Best practice sending BNB is to use call method	● Informational
STATE-1	Flipping state for investmentStatus could be adjusted	● Informational

PRAC-1	Best practice to follow check effects interactions within functions	● Informational
SEND-2	Assignment of msgSender() to user variable uses unnecessary gas	● Informational
CHECK-1	Overflow checks can be removed as Solidity 0.8.0+ automatically checks	● Informational
SAFE-1	SafeMath does not need to be imported for projects using Solidity 0.8.0+	● Informational
CALC-1	Update payouts could be negative from calculation	● Medium

SEND-1 | Do not use msgSender() unless implementing EIP-2771

Description

The `msgSender()` from Context is being used to fetch the `msg.sender` for the caller across a series of functions. Unless meta transactions (EIP-2771) are planned to be used for the contract then this should be removed and `msg.sender` should be used instead.

There is a modifier for `onlyEOA` used across a series of functions checking that the `tx.origin` is `msg.sender`, and in all of these cases there would be no need to use `msgSender()`.

Recommendation

Remove `msgSender()` and use `msg.sender` unless `metaTransaction` are planned to be implemented into this contract.

PACK-1 | Struct packing could be improved for Stats

Description

The struct for Stats includes a series of variables that are unlikely to exceed the max value of `uint128` ($3.4028237e+38$) such as `invested`, `reinvested`, `withdrawn`, and `taxes`. The values for `xInvested`, `xReinvested`, and `xWithdrawn` are all iterated on by 1 as users interact with the product. It is fair to assume that the number of times that a user interacts in each of these ways is likely to be less than the max value of `uint64` or `uint32`, meaning these variables could be packed in this way. Packing the struct more efficiently will reduce gas costs as users interact with the protocol.

Recommendation

Reduce the size of the integers from `uint256` to smaller sizes that are unlikely to overflow with usage.

VAR-1 | State variables iterated on but not used for functionality could be removed

Description

State variables that are being iterated but are not core to the functionality of the product can be removed. If the variables served the purpose of updating the front-end, a more efficient approach could be to use the events that are currently being emitted with a tool such as The Graph to show up to date information on the front-end.

This logic could be applied to the variables in Stats such as [xInvested](#), [xReinvested](#) and [xWithdrawn](#), and state variables such as [totalTx](#)s. It will reduce the amount of gas that is being emitted by reducing the amount of times state variables are updated in function.

UINT-1 | MAX_UINT state variable is not used

Description

The state variable MAX_UINT is initialized but its value is not used through the contract meaning that it can be removed.

Recommendation

Remove the state variable for MAX_UINT if it is not being used.

VAR-2 | State variables in DataSets could be packed into fewer slots

Description

Similar to the previous discussion on packing state variables, the variables in Datasets could use smaller uint values to be packed more efficiently. This applies more specifically to variables that are often fetched from storage in the same function, which will reduce the amount of gas being spent to load.

Recommendation

Pack variables in datasets into smaller uint values that are being fetched in the same function to reduce gas cost e.g. [lockedBalance](#) and [DividendBalance](#) or [lastPayout](#). The number of state variables being occupied in succession from previously declared variables must be calculated to ensure they will be in the same slot.

METH-1 | Best practice for sending BNB is to use call method

Description

The best practice approach to send native tokens (ETH/ BNB) as per the Solidity docs is to use the call method. It would be recommended to adopt the suggested approach to follow best practice in the [recoverBNB](#) function when sending BNB.

Recommendation

Use the best practice approach to send BNB in the recoverBNB function:

```
(bool sent, bytes memory data) = to.call{value: msg.value}("");  
require(sent, "BNB_TX_FAIL");
```

STATE-1 | Flipping state for investmentStatus could be adjusted

Description

The state setter for `investmentStatus` currently will only set to true, which may be the desired functionality for the product. However, if there are instances where `investmentStatus` should be set to false to pause contract interactions then the function could set the value to the opposite bool.

Recommendation

Set the bool to the opposite value in `changeInvestmentStatus()` function if there are instances where the status should be paused again:

```
function changeInvestmentStatus() external onlyOwner {  
    investmentStatus = !investmentStatus;  
}
```

PRAC-1 | Best practice to follow check effects interactions within functions

Description

Although non-reentrant is being used on functions such as reward, invest, `investFor`, and reinvest, it would still be recommended to follow best practice with check-effects-interactions. This would mean the state variables are updated before external calls i.e. in the example of invest `totalDeposits` will be added to then `investing()` will be called assigning the return value to a named `uint256` variable that is return, and finally `transferFrom` is called

Recommendation

Follow best practice of check-effects-interactions by updating state variables first, calling internal functions next, and finally calling external contracts last.

```
function invest(uint256 amount) external <modifiers> returns (uint256
value) {
    require(amount >= minInvest, "min invest amount error");
    totalDeposit += amount;
    value = investing(msg.sender, amount);
    mtoken.transferFrom(msg.sender, address(this), amount);
}
```

SEND-2 | Assignment of msgSender() to user variable uses unnecessary gas

Description

The assignment of `msgSender()` to a memory variable will always use more gas than using the global variable of `msg.sender`. It would be recommended to use `msg.sender` in all cases where this happens to improve gas efficiency, assuming the implementation of meta transactions is not used for this contract.

This occurs in the following functions: `invest()`, `reinvest()`, `withdraw()`, and `exit()`.

Recommendation

Do not assign `msg.sender` to a memory variable

CHECK-1 | Overflow checks can be removed as Solidity 0.8.0+ automatically checks

Description

The solidity compiler will automatically check for over and underflows in mathematical operation when using version 0.8.0+. This means, there is no requirement to check whether the addition of variables is safe in required statements as this will happen by default.

Recommendation

Remove the require statements checking for integer overflow.

SAFE-1 | SafeMath does not need to be imported for projects using Solidity 0.8.0+

Description

As Solidity 0.8.0+ checks for over and under flows with mathematical operations, it means there would be no need to use SafeMath on integer values. The safe math checks would be automatically baked into all calculations that are being completed by the compiler.

Recommendation

Remove the usage of the SafeMath library to complete mathematical operations.

CALC-1 | Update payouts could be negative from calculation

Description

The calculation being used to subtract from `payoutsTo[user]` in the exit function deducts `_updatedPayouts` from the existing value. However, the `_updatedPayouts` value is the combination of the amount of tokens being withdrawn and the addition of the `taxedAmount` multiplied by the magnitude.

The `payoutsTo` field is updated by adding `profitPerShare` multiplied by `amountOfTokens` in `investToken` field and adding dividends multiple by magnitude elsewhere. Each of these values stem from the base of `amountOfTokens` and in the case of dividends the outstanding `profitPerShare` based on the user balance readjusted by magnitude.

In practice, if the user were to withdraw a large amount of tokens close to their total balance then the calculation would deduct 100% of the withdrawal balance plus (10% of withdraw * magnitude) and deduct this from the `payoutsTo` mapping field. There is a risk that the value being deducted could be more than the `payoutsTo` value leaving the mapping in negative territory. As the value for `payoutsTo` is an `int256` there is no risk of underflow but there could be implications on usage if the payout balance from an address reaches negative territory.

Example: user depositing 100 XYZ token

(Assuming values for token are 18 decimals)

Step - Invest function called with 100 XYZ

- 90% of the deposit is used for amountOfTokens
- Balance of 0 for tokenBalanceLedger becomes 90e18
- Payouts is adjusted to prevent the user claiming 90e18 worth of rewards at 1 per share
- $\text{payoutsTo} \text{ equals } 0 \text{ += } 90\text{e}18 * 1 = 90\text{e}18$

Step - Exit called for 80 XYZ

- Withdrawing 80e18 tokens
- Undivided dividends equals 8e18 (10% of 80e18)
- taxedAmount is 72e18 (90% of 80e18)
- LedgerBalance is adjusted correctly
- _updatePayouts equals $((1 * 80\text{e}18) + (72\text{e}18 * 10^{**}10)) = 7.2\text{e}+29$
- payoutsTo mapping currently equals 90e18
- Deducting updatePayouts from payoutsTo would lead to negative territory as $7.2\text{e}+29 > 90\text{e}18$

Recommendation

Adjust the calculation for the payout adjustment to reduce the likelihood of overflow. It is difficult to suggest an exact solution without a total understanding of the purpose of magnitude (See notes).

Additional Notes

The limiting factor to commenting on some of the mathematical operations in the contract is understanding the usage of the magnitude state variables (10^{**10}). The values for dividends are often multiplied by magnitude to updated `payoutsTo` and there is also usage to update `profitPerShare`.

To fully comment on whether the math is being calculated in the preferred manner, it would be useful to hear the developer's comments on the purpose of magnitude across the board.

DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement.

This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without [Safetin's](#) prior written consent. This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts [Safetin](#) to perform a security assessment.

This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance. This report should not be used in any way

Safetin security assessment to make decisions around investment or involvement with any particular project.

This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Safetin's position is that each company and individual are responsible for their own due diligence and continuous security. Safetin's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or fun.