PRISMA SHIELD

DEEP LOGIC AUDIT REPORT

Ooze Finance Immutable Contracts

JUN 21 2022

PRISMA SHIELD

Table of Contents

Overview

This audit includes the following user-interactable contracts:

- OozeToken
- OozeTaxVault
- BarrelToken
- OozeLiquidityVault
- SlimeToken

Note: This audit does not include the PreLaunchMinter. Interact with that contract at your own risk.

Project Summary

Project Name	Ooze Finance
Network	Fantom
Language	Solidity

What is a Deep Logic Audit?

A deep logic smart contract audit is a human-driven code review that checks all of the code business logic for bugs, mathematical errors, and security risks. The audit verifies that the code honors the whitepaper. In addition, this service includes mainnet testing and proactive communication with the project owners to ensure full comprehension of the project to provide the best possible code review quality.

Overview

Findings Summary

		Findings Resolved
Total Findings	16	16
High Security Findings	0	0
Medium Security Findings	3	3
High Logical Findings	1	1
Medium Logical Findings	2	2
Informational Findings	10	10

Overview

Findings Summary

ID	Section	Type	Severity	Page	Status
ERC-01	ERC20Taxable	Logical	Informational	9	Resolved
ERC-02	ERC20Taxable	Logical	Informational	10-11	Resolved
ERC-03	ERC20Taxable	Logical	Informational	12	Resolved
VLT-01	OozeTaxVault	Security	Medium	16	Resolved
STR-01	EncodedStrings	Logical	Informational	18	Resolved
STR-02	EncodedStrings	Logical	Informational	19	Resolved
BRL-01	BarrelToken	Logical	Informational	23	Resolved
BRL-02	BarrelToken	Logical	Informational	24	Resolved
LIQ-01	OozeLiquidityVault	Security	Medium	27-28	Resolved
LIQ-02	OozeLiquidityVault	Security	Medium	29-30	Resolved
SLM-01	SlimeToken	Logical	High	32	Resolved
SLM-02	SlimeToken	Logical	Medium	33	Resolved

Overview

Findings Summary

ID	Section	Type	Severity	Page	Status
SLM-03	SlimeToken	Logical	Medium	34	Resolved
SLM-04	SlimeToken	Logical	Informational	35	Resolved
SLM-05	SlimeToken	Logical	Informational	36-37	Resolved
SLM-06	SlimeToken	Logical	Informational	38-39	Resolved

Contract Addresses

Please ensure you are interacting with the correct contract addresses.

BarrelToken

0xf45439A5f85a41Fc2D626e80fA44FB5A2DD4fF6B

Please ensure the following in the contract before any NFT transfers:

1. Open https://ftmscan.com/address/0xf45439A5f85a41Fc2D626e80fA44FB5A2DD4fF6B#readContract
2. Ensure that the "sewerContractAddress" field matches the contract address of the "Sewer" contract

OozeToken

0x60e6AFEb3ac2fBc82A8d312BEa3B47DC6b4848d2

Please ensure the following in the contract:

1. Open https://ftmscan.com/address/0x60e6afeb3ac2fbc82a8d312bea3b47dc6b4848d2#readContract
2. Ensure that the "vaultAddress" field is 0xab35de44b394db86de7f720a5a5ddec458949d11

OozeTaxVault

0xaB35De44b394DB86dE7f720a5A5dDEC458949D11

Please ensure the following in the contract:

- Open https://ftmscan.com/address/0xaB35De44b394DB86dE7f720a5A5dDEC458949D11#readContract
- Ensure that the "oozeAddress" field is 0x60e6afeb3ac2fbc82a8d312bea3b47dc6b4848d2

Contract Addresses

OozeLiquidityVault
0x981b2641d2E685130035953c3C69Ba2a023ac3c7

Please ensure the following in the contract:

- Open https://ftmscan.com/address/0x981b2641d2E685130035953c3C69Ba2a023ac3c7#readContract
- Ensure that the "slimeAddress" field is 0xf00a4e49edb40c2483fc6d398694d619d44fa182

SlimeToken
0xF00A4e49Edb40C2483fC6d398694d619D44FA182

Please ensure the following in the contract:

- Open https://ftmscan.com/address/0xF00A4e49Edb40C2483fC6d398694d619D44FA182#readContract
- Ensure that the "oozeTaxVault" field is 0xab35de44b394db86de7f720a5a5ddec458949d11

Audit Findings

ERC20Taxable

ERC-01 - Logical Informational Severity

The "_internalSender" and "_internalRecipient" variables are "private" and don't have any getter functions to view their values, which will limit visibility into the contract and make debugging issues difficult.

Recommendation

Add getter functions for these variables:

```
function isSenderTaxExempt(address addr) external view returns (bool) {
    return _internalSender[addr];
}

function isRecipientTaxExempt(address addr) external view returns (bool) {
    return _internalRecipient[addr];
}
```

Resolution

The team has resolved this issue.

The variables have been changed to "public".

Audit Findings

ERC20Taxable

ERC-02 - Logical Informational Severity

Multiplying the "defaultTaxRate" by 1e12 has no effect on the accuracy of the result, and can be removed to save on gas.

Recommendation

Change the tax math to the following:

```
function calculateTransferTaxRate(address _from, address _to) public view returns (uint256 taxRate) {
    taxRate = 0;

    if (
        vaultAddress != address(0) &&
        _to != vaultAddress &&
        !_internalRecipient[_to] &&
        !_internalSender[_from]
    )
    {
        taxRate = uint256(defaultTaxRate);
    }
}

function calculateTransferTax(
    address _from,
    address _to,
    uint256 _value
) public view returns (uint256 adjustedValue, uint256 taxAmount) {
    adjustedValue = _value;
    taxAmount = 0;

    uint256 taxRate = calculateTransferTaxRate(_from, _to);
    if (taxRate > 0) {
        taxAmount = (_value * taxRate) / 100;
        adjustedValue -= taxAmount;
    }
}
```

Audit Findings

ERC20Taxable

ERC-02 - Logical Informational Severity

Resolution

The team has resolved this issue.

The 1e12 multiplier has been removed from the maths.

Audit Findings

ERC20Taxable

ERC-03 - Logical Informational Severity

"_transfer" can be overriden instead of overriding both "transfer" and "transferFrom", reducing and simplifying the code.

Recommendation

Rename "_taxableTransfer" to "_transfer" and set it to "internal override". Delete the overriden "transfer" and "transferFrom" function overrides:

```
function _transfer(
    address from,
    address to,
    uint256 amount
) internal override {
    (uint256 adjustedValue, uint256 taxAmount) = calculateTransferTax(_from, _to, _amount);

    if (taxAmount > 0) {
        super._transfer(_from, vaultAddress, taxAmount);
        emit TaxPaid(_from, vaultAddress, taxAmount);
    }
    super._transfer(_from, _to, adjustedValue);
}
```

Resolution

The team has resolved this issue.

The "_transfer" function has been overriden instead of "transfer" and "transferFrom".

Overview

ERC20Taxable

- This abstract contract adds a tax mechanism on the transfers of ERC20 tokens.
- The "defaultTaxRate" publicly viewable variable contains the percentage of tokens that will be deducted in tax, and sent to the "vaultAddress". It is limited to 10%. The contract owner has admin powers to change this value.
- The "vaultAddress" publicly viewable variable contains the address that receives the taxes from the transfers. This is meant to be the "OozeTaxVault" contract address. The contract owner has admin powers to change this address.
- The "internalSender" and "internalRecipient" publicly viewable variables contain the addresses that are exempt from taxes. The contract owner has admin powers to change these values.
- The "calculateTransferTax" function calculates the amount of tax that should be deducted from transfers.
- The "_transfer" function sends the taxes to the appropriate address, and sends the rest to the destination address of the transfer.

Audit Findings

OozeToken

No findings

Overview

OozeToken

- This contract implements "ERC20Taxable", adding the ability to mint new tokens, as well as the ability to pause token transfers and mints.

- The contract owner has admin powers to mint new tokens, as well as the ability to pause and unpause token transfers and mints.

Audit Findings

OozeTaxVault

VLT-01 - Security Medium Severity

The "withdrawTokens" and "sendTokens" functions allow for the interaction with any contract whose address is passed as an argument in the "tokenAddress" parameter. This could result in interacting with malicious contracts.

Recommendation

Since the "OozeTaxVault" is meant to only hold and send Ooze tokens, it would be best to lock down the "OozeToken" contract address, either by passing it once in the "OozeTaxVault" constructor, or by having an "onlyRole(DEFAULT_ADMIN_ROLE)" setter function for the "OozeToken" contract address.

```
IERC20 public ooze;

constructor(address owner, address oozeAddr) {
    _grantRole(DEFAULT_ADMIN_ROLE, owner);
    ooze = IERC20(oozeAddr);
}

function withdrawTokens(uint256 amount) public onlyRole(WITHDRAW_ROLE) {
    require(ooze.transfer(msg.sender, amount), "Withdrawal failed");
}

function sendTokens(uint256 amount, address destination) public onlyRole(WITHDRAW_ROLE) {
    require(ooze.transfer(destination, amount), "Withdrawal failed");
}
```

Resolution

The team has resolved this issue.

The functions can only call the "OozeToken" contract, which is set in the constructor.

Overview

OozeTaxVault

- This contract is meant to receive and hold Ooze tokens, while allowing other allow-listed contracts to withdraw Ooze tokens from it.
- The "withdrawTokens" function allows addresses with the "WITHDRAW_ROLE" to withdraw Ooze tokens held in the "OozeTaxVault".
- The "sendTokens" function allows addresses with the "WITHDRAW_ROLE" to send Ooze tokens held in the "OozeTaxVault" to any address.

Audit Findings

EncodedStrings

STR-01 - Logical Informational Severity

The gas cost and readability of the "bytes32ToString" function can be improved by directly using "mstore(dest, self)".

Recommendation

Use "mstore(dest, self)":

```
function bytes32ToString(bytes32 self) internal pure returns (string memory) {
    string memory ret = new string(len(self));
    assembly {
        let dest := add(ret, 0x20)
        mstore(dest, self)
    }
    return ret;
}
```

Resolution

The team has resolved this issue.

The recommended code change has been applied.

Audit Findings

EncodedStrings

STR-02 - Logical Informational Severity

The "encode" function limits the byte size to < 32. However, the byte length could be allowed to go up to <= 32, and the code would still function as expected.

Recommendation

Allow the byte length to go up to <= 32:

```
require(b.length > 0 && b.length <= 32, "INVALID STRING LENGTH");
```

Resolution

The team has resolved this issue.

The byte length is allowed to go up to 32.

Overview

EncodedStrings

- This library provides functions that can encode string data types into bytes32 data types, and decode bytes32 data types into string data types.
- The "encode" function takes a string argument and converts it into the corresponding bytes32 data type.
- The "len" function receives a bytes32 argument that is representing a string, and returns the number of bytes of that string, between 0 and 32 inclusive.
- The "bytes32ToString" function receives a bytes32 argument that is representing a string, and decodes it into the corresponding string.

Audit Findings

ERC2981Global

No findings

Overview

ERC2981Global

- The ERC2981 abstract contract implements the EIP-2981 NFT royalty standard. In short, that standard simply requires implementing the "royaltyInfo" function, which can be used by supporting NFT marketplaces to retrieve the royalty payment information for NFTs. Royalty payment denotes how much of the NFT sale price goes to the NFT creator.

- The "supportsInterface" publicly viewable function returns true if this contract implements the interface defined by "interfaceId" argument.

- The "royaltyInfo" publicly viewable function returns how much royalty of the sale price of NFTs is owed and to whom it is owed. The same percentage is used to calculate the royalty of all the NFTs.

Audit Findings

BarrelToken

BRL-01 - Logical Informational Severity

The contract owner is not meant to be able to mint new NFTs. Only other contracts like the "PreLaunchMinter" and the "Sewer" should be able to do that.

Recommendation

Delete "_grantRole(MINTER_ROLE, owner)".

Resolution

The team has resolved this issue.

The contract owner no longer has the "MINTER_ROLE" on contract deployment.

Audit Findings

BarrelToken

BRL-02 - Logical Informational Severity

The "supportsInterface" function should return true for the "IBarrelToken" interface since the "BarrelToken" contract implements it.

Recommendation

Modify the "supportsInterface" function to return true for the "IBarrelToken" interface:

```
return interfaceId == type(IBarrelToken).interfaceId || super.supportsInterface(interfaceId);
```

Resolution

The team has resolved this issue.

The function returns true for the "IBarrelToken" interface.

Overview

BarrelToken

- The "BarrelToken" contract implements the NFTs used in the Ooze Finance project. In more technical terms, the contract implements the EIP-721: Non-Fungible Token Standard. It also inherits from "ERC2981Global", adding royalty information to NFT transfers. Each NFT is assigned a unique name.

- The "sewerContractAddress" publicly viewable variable contains the address of the "Sewer" contract. The contract owner has admin powers to change this address.

- The "encodedTokenName" publicly viewable variable maps each NFT token ID to its corresponding unique encoded name.

- The "features" publicly viewable variable maps each NFT token ID to some of its metadata (e.g. clan, is the user a moderator or dev, etc.) encoded in a bitmap.

- The "tokenByName" publicly viewable variable maps each unique encoded name to its corresponding NFT token ID.

- The "safeMint" function is used to mint a new NFT, assigning it a features bitmap and a unique name. This function is meant to only be called by other contracts like the "PreLaunchMinter" and the "Sewer".

- The "locateTokenByName" publicly viewable function returns the token ID corresponding to the specified "name" argument.

Overview

BarrelToken

- The "tokenName" publicly viewable function returns the unique name corresponding to the specified "tokenId" argument.
- The "setBaseURI" function is used to set the base URI that, when appended with token IDs, returns their corresponding NFT metadata (e.g. NFT picture) in a JSON object. The contract owner has admin powers to call this function.
- The "pause" and "unpause" functions are used to disable/enable NFT token transfers. The contract owner has admin powers to call these functions.
- The "_beforeTokenTransfer" function calls the "applyNftTransferTax" function in the "Sewer" contract before any NFT transfer.
- The "setDefaultRoyalty" function is used to set the royalty settings for what percentage should be deducted in royalty from NFT buys and sells, and which address the royalty should be sent to. The contract owner has admin powers to call this function.
- The "supportsInterface" publicly viewable function returns true if this contract implements the interface defined by "interfaceId" argument.

Audit Findings

OozeLiquidityVault

LIQ-01 - Security Medium Severity

The "withdrawTokens" and "sendTokens" functions allow for the interaction with any contract whose address is passed as an argument in the "tokenAddress" parameter. This could result in interacting with malicious contracts.

Recommendation

Since the "OozeLiquidityVault" is meant to only hold and send Slime tokens, it would be best to lock down the "SlimeToken" contract address, either by passing it once in the "OozeLiquidityVault" constructor, or by having an "onlyRole(DEFAULT_ADMIN_ROLE)" setter function for the "SlimeToken" contract address.

```
address public slimeAddress;
IERC20 private slime;

constructor(address owner, address slimeAddr) {
    _grantRole(DEFAULT_ADMIN_ROLE, owner);
    _grantRole(WITHDRAW_ROLE, owner);
    slimeAddress = slimeAddr;
    slime = IERC20(slimeAddr);
    emergencyWithdrawalStartTime = type(uint256).max;
}

function withdrawTokens(uint256 amount) external onlyRole(WITHDRAW_ROLE) emergencyWithdrawalTimeElapsed {
    require(slime.transfer(msg.sender, amount), "Withdrawal failed");
    emit Withdrawal(msg.sender, slimeAddress, amount);
}

function sendTokens(uint256 amount, address destination)
    external onlyRole(WITHDRAW_ROLE) emergencyWithdrawalTimeElapsed
{
    require(slime.transfer(destination, amount), "Withdrawal failed");
    emit Withdrawal(destination, slimeAddress, amount);
}
```

Audit Findings

OozeLiquidityVault

LIQ-01 - Security Medium Severity

Resolution

The team has resolved this issue.

The functions can only call the "SlimeToken" contract, which is set in the constructor.

Audit Findings

OozeLiquidityVault

LIQ-02 - Security Medium Severity

After a week passing from calling "beginEmergencyWithdraw", the contract owner is allowed to withdraw any amount of Slime tokens from the contract for any number of times until "cancelEmergencyWithdraw" is called.

Recommendation

1. Specify the amount intended for withdrawal in the "beginEmergencyWithdraw", and only allow that amount or less to be withdrawn
2. Reset the timer in the "withdrawTokens" and "sendTokens" functions

```
uint256 public emergencyAmountToWithdraw;

event EmergencyWithdrawalStarted(uint256 amount);

function beginEmergencyWithdraw(uint256 amount) external onlyRole(WITHDRAW_ROLE) {
    // Require emergency withdrawal not already started
    require(emergencyWithdrawalStartTime == type(uint256).max);
    emergencyWithdrawalStartTime = block.timestamp;
    emergencyAmountToWithdraw = amount;
    emit EmergencyWithdrawalStarted(amount);
}

function withdrawTokens(uint256 amount) external onlyRole(WITHDRAW_ROLE) emergencyWithdrawalTimeElapsed {
    require(amount <= emergencyAmountToWithdraw, "Amount larger than max withdrawal allowance");
    emergencyWithdrawalStartTime = type(uint256).max;
    require(slime.transfer(msg.sender, amount), "Withdrawal failed");
    emit Withdrawal(msg.sender, slimeAddress, amount);
}

function sendTokens(uint256 amount, address destination)
    external onlyRole(WITHDRAW_ROLE) emergencyWithdrawalTimeElapsed
{
    require(amount <= emergencyAmountToWithdraw, "Amount larger than max withdrawal allowance");
    emergencyWithdrawalStartTime = type(uint256).max;
    require(slime.transfer(destination, amount), "Withdrawal failed");
    emit Withdrawal(destination, slimeAddress, amount);
}
```

Audit Findings

OozeLiquidityVault

LIQ-02 - Security Medium Severity

Resolution

The team has resolved this issue.

The maximum number of Slime tokens to be withdrawn has to be specified a week ahead of the actual withdrawal, and the withdrawal itself resets the timer.

Overview

OozeLiquidityVault

- This contract is meant to hold Slime tokens. Withdrawing Slime tokens from this contract requires a 1 week notice through calling the "beginEmergencyWithdraw" function before being able to actually perform the withdrawal. The amount to be withdrawn has to be specified when calling "beginEmergencyWithdraw".

- The "beginEmergencyWithdraw" function starts a timer, where once it exceeds one week, Slime tokens are allowed to be withdrawn from the contract. The contract owner has admin powers to call this function.

- The "cancelEmergencyWithdraw" function resets and stops the timer. The contract owner has admin powers to call this function.

- The "withdrawTokens" function allows addresses with the "WITHDRAW_ROLE" to withdraw Slime tokens held in the "OozeLiquidityVault" only after one week has passed since calling the "beginEmergencyWithdraw" function, and without the "cancelEmergencyWithdraw" function being called. The timer is reset after this function is called. The contract owner has admin powers to call this function.

- The "sendTokens" function allows addresses with the "WITHDRAW_ROLE" to send Slime tokens held in the "OozeLiquidityVault" to any address only after one week has passed since calling the "beginEmergencyWithdraw" function, and without the "cancelEmergencyWithdraw" function being called. The timer is reset after this function is called. The contract owner has admin powers to call this function.

Audit Findings

SlimeToken

SLM-01 - Logical High Severity

The "collateralToTokenSwapOutput" function does not verify that "collateralSold" <= "maxCollateral". This can result in users paying more than the maximum they've specified.

Recommendation

Require that "collateralSold" <= "maxCollateral":

```
function collateralToTokenSwapOutput(uint256 tokensBought, uint256 maxCollateral)
    public whenNotPaused returns (uint256)
{
    ...
    uint256 collateralSold = getOutputPrice(tokensBought, collateralReserve, tokenReserve);
    require(collateralSold <= maxCollateral);
    ...
}
```

Resolution

The team has resolved this issue.

The amount of collateral tokens to be sold has to be less than the maximum amount specified by the user.

Audit Findings

SlimeToken

SLM-02 - Logical Medium Severity

The "setEarlySellTax" function can be called at any time, regardless of whether or not the early sell tax period has already started.

Recommendation

Only allow "setEarlySellTax" to be called before the early sell tax period has begun:

```
function setEarlySellTax(
    uint8 tax,
    uint256 start,
    uint256 duration
) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(tax <= 40, "Invalid Early Sell tax amount");
    require(start > block.timestamp, "Early tax period has to start in the future");
    require(
        earlyTaxStart == 0 || block.timestamp < earlyTaxStart,
        "Early tax start can't be changed after it has already started"
    );
    earlySellTax = tax;
    earlyTaxStart = start;
    earlyTaxDuration = duration;
}
```

Resolution

The team has resolved this issue.

The function can only be called before the early sell tax period has begun.

Audit Findings

SlimeToken

SLM-03 - Logical Medium Severity

Assuming that the ERC-02 finding has been applied, the "calculateSalesTax" function should be changed to multiply the result of the "token.calculateTransferTaxRate" call by 1e12 such that the math remains correct.

Recommendation

Multiply the result of the "token.calculateTransferTaxRate" call by 1e12:

```
function calculateSalesTax(address seller)
    public
    view
    returns (
        uint256 taxRate1e12,
        uint256 extraTax,
        uint256 marketingTax
    )
{
    taxRate1e12 = token.calculateTransferTaxRate(seller, address(this)) * 1e12;
    ...

}
```

Resolution

The team has resolved this issue.

The suggested math change has been applied.

Audit Findings

SlimeToken

SLM-04 - Logical Informational Severity

Since it is already known that the collateral address is that of Usdc before contract deployment, it is better to simply hardcode the Usdc contract address in the code as opposed to setting it in the constructor.

Recommendation

Hardcode the Usdc contract address (0x04068DA6C83AFCFA0e13ba15A6696662335D5B75) in the code and don't set it in the constructor:

```
address public constant COLLATERAL_ADDRESS = 0x04068DA6C83AFCFA0e13ba15A6696662335D5B75;
IERC20 private constant COLLATERAL_TOKEN = IERC20(COLLATERAL_ADDRESS);

constructor(
    address owner,
    address oozeTokenAddress
) ERC20("SlimeToken", "SLIME") {
    _grantRole(DEFAULT_ADMIN_ROLE, owner);
    _grantRole(PAUSER_ROLE, owner);
    _grantRole(MINTER_ROLE, owner);

    tokenAddress = oozeTokenAddress;
    token = IERC20Taxable(oozeTokenAddress);

    lastBalanceTime = block.timestamp;
}
```

Resolution

The team has resolved this issue.

The Usdc address has been hardcoded into the code and cannot be changed.

Audit Findings

SlimeToken

SLM-05 - Logical Informational Severity

The math in "_getTokenToCollateralInputPriceWithTax" could result in minor inaccuracies due to integer division where the sum of the tax vault and marketing taxes don't add up exactly to the total tax deducted in the transaction.

Recommendation

Change the math such that the total of the different taxes matches exactly the deducted tax:

```
function _getTokenToCollateralInputPriceWithTax(uint256 tokensSold, address seller)
    private
    view
    returns (
        uint256 collateralOutput,
        uint256 toVault,
        uint256 toMarketingWallet
    )
{
    ...
    if (taxRate1e12 > 0) {
        uint256 taxAmount = (tokensSold * taxRate1e12) / 100e12;

        effectiveTokensSold -= taxAmount;

        // This is exactly the same amount of transfer tax to be deducted
        // in the Ooze token transfer.
        uint256 defaultTaxRate = taxRate1e12 - extraTaxRate - marketingTax;
        uint256 defaultTaxAmount = defaultTaxRate * tokensSold / 100e12;

        toMarketingWallet = marketingTax * tokensSold / 100e12;

        if (extraTaxRate > 0) {
            toVault = taxAmount - defaultTaxAmount - toMarketingWallet;
        } else {
            toMarketingWallet = taxAmount - defaultTaxAmount;
        }
    }
    ...
}
```

Audit Findings

SlimeToken

SLM-05 - Logical Informational Severity

Resolution

The team has resolved this issue.

The suggested math change has been applied.

Audit Findings

SlimeToken

SLM-06 - Logical Informational Severity

The math in "_getTokenToCollateralOutputPriceWithTax" could result in minor inaccuracies due to integer division where the sum of the tax vault and marketing taxes don't add up exactly to the total tax deducted in the transaction.

Recommendation

Change the math such that the total of the different taxes matches exactly the deducted tax:

```
function _getTokenToCollateralOutputPriceWithTax(uint256 collateralOutput, address seller)
    private
    view
    returns (
        uint256 tokensSold,
        uint256 toVault,
        uint256 toMarketingWallet
    )
{
    ...
    if (taxRate1e12 > 0) {
        tokensSold = (baseTokensSold * 100e12) / (100e12 - taxRate1e12);
        uint256 taxAmount = tokensSold - baseTokensSold;

        // This is exactly the same amount of transfer tax to be deducted
        // in the Ooze token transfer.
        uint256 defaultTaxRate = taxRate1e12 - extraTaxRate - marketingTax;
        uint256 defaultTaxAmount = defaultTaxRate * tokensSold / 100e12;

        toMarketingWallet = marketingTax * tokensSold / 100e12;

        if (extraTaxRate > 0) {
            toVault = taxAmount - defaultTaxAmount - toMarketingWallet;
        } else {
            toMarketingWallet = taxAmount - defaultTaxAmount;
        }
    }
}
```

Audit Findings

SlimeToken

SLM-06 - Logical Informational Severity

Resolution

The team has resolved this issue.

The suggested math change has been applied.

Overview

SlimeToken

- This contract is the DEX used to exchange between Ooze and Usdc. Liquidity of Ooze and Usdc can be added to it in exchange for the Slime LP token. The Slime tokens can be exchanged back to the corresponding Ooze and Usdc liquidity.

- The "tokenAddress" publicly viewable variable (and corresponding "token" variable) are supposed to have the "OozeToken" contract address. These variables are set once in the constructor.

- The "earlySellTax", "earlyTaxStart", and "earlyTaxDuration" publicly viewable variables are used to add extra sell tax for a set period of time. The extra tax declines each hour linearly over that period. The "earlySellTax" has a default value of 40, which is the maximum it can be set to. This means that the total sell tax can go up to a maximum of 50% (40% early sell tax + 10% default transfer tax). The contract owner has admin powers to change these values.

- The "marketingShare" and "marketingWallet" publicly viewable variables decide the share of the taxes during the extra early sell tax period that is supposed to go to the marketing wallet. The "marketingWallet" variable is meant to be set to the multisig address. The "marketingShare" has a default value of 30, and can be set to a maximum of 40. It represents the percentage of the total tax percentage that goes to the marketing wallet (e.g. 40% * 50% = 20%). The resulting percentage cannot exceed the extra sell tax percentage. The contract owner has admin powers to change these values.

- The "oozeTaxVault" publicly viewable variable is where the extra early sell taxes are sent (apart from those sent to the marketing wallet). This is meant to be set to the "OozeTaxVault" contract address. The contract owner has admin powers to change this value.

- The "decimals" publicly viewable function denotes the number of decimal places of the Slime token, which is six, matching that of Usdc.

Overview

SlimeToken

- The "getInputPrice" publicly viewable function returns the amount of tokens B bought if an "inputAmount" of token A was sold. This is based on the ratio of tokens in the liquidity pool. This uses the standard Uniswap "getAmountOut" function, and charges an 0.3% liquidity fee.

- The "getOutputPrice" publicly viewable function returns the amount of tokens B sold if an "outputAmount" of token A was bought. This is based on the ratio of tokens in the liquidity pool. This uses the standard Uniswap "getAmountIn" function, and charges an 0.3% liquidity fee.

- The "collateralToTokenSwapInput" function is used to sell an exact amount of Usdc to buy the corresponding Ooze based on the price. The contract owner has admin powers to pause this function.

- The "collateralToTokenSwapOutput" function is used to buy an exact amount of Ooze to sell the corresponding Usdc based on the price. The contract owner has admin powers to pause this function.

- The "tokenToCollateralSwapInput" function is used to sell an exact amount of Ooze to buy the corresponding Usdc based on the price. Taxes are deducted from the Ooze being sold, and the Usdc price is calculated on the remaining amount. The contract owner has admin powers to pause this function.

- The "tokenToCollateralSwapOutput" function is used to buy an exact amount of Usdc to sell the corresponding Ooze based on the price. Extra Ooze tokens on top of those corresponding to the USDC price are paid by the user for taxes. The contract owner has admin powers to pause this function.

- The "getReserves" publicly viewable function returns the amount of Ooze and Usdc in the contract used for the liquidity pool respectively.

Overview

SlimeToken

- The "calculateSalesTax" publicly viewable function returns all of the tax percentages when Ooze is being sold by the specified "seller" address.
- The "getCollateralToTokenInputPrice" publicly viewable function returns the amount of Ooze bought if the specified "collateralSold" amount of Usdc was sold.
- The "getCollateralToTokenOutputPrice" publicly viewable function returns the amount of Usdc sold if the specified "tokensBought" amount of Ooze was bought.
- The "getTokenToCollateralInputPrice" publicly viewable function returns the amount of Usdc bought if the specified "tokensSold" amount of Ooze was sold. This does not account for the Ooze taxes.
- The "getTokenToCollateralOutputPrice" publicly viewable function returns the amount of Ooze sold if the specified "collateralBought" amount of Usdc was bought. This does not account for the Ooze taxes.
- The "getTokenToCollateralInputPriceWithTax" publicly viewable function returns the amount of Usdc bought if the specified "tokensSold" amount of Ooze was sold. This accounts for the Ooze taxes.
- The "getTokenToCollateralOutputPriceWithTax" publicly viewable function returns the amount of Ooze sold if the specified "collateralOutput" amount of Usdc was bought. This accounts for the Ooze taxes.
- The "addLiquidity" function is used to add Ooze and Usdc liquidity in exchange for newly minted Slime LP tokens. Only addresses with the "MINTER_ROLE" or "DEFAULT_ADMIN_ROLE" can call this function. The contract owner has admin powers to pause this function. The contract owner has admin powers to call this function even when the contract is paused.

Overview

SlimeToken

- The "removeLiquidity" function is used to burn Slime LP tokens to take out of the contract LP the corresponding Ooze and Usdc tokens. Only addresses with the "MINTER_ROLE" or "DEFAULT_ADMIN_ROLE" can call this function. The contract owner has admin powers to pause this function. The contract owner has admin powers to call this function even when the contract is paused.

- The contract owner has admin powners to pause Slime token transfers. The contract owner can transfer Slime tokens even when the contract is paused.

Audit Findings

Additional Findings

IERC20Taxable
No findings

IOozeToken
No findings

IVault
No findings

IBarrelToken
No findings

INftTransferTax
No findings

ILiquidityVault
No findings

IOozeSwap
No findings

Audit Findings

Additional Findings

Sewer

No findings

Note: Only the single function "applyNftTransferTax" that is relevant to the other contracts in this audit has been audited.

How to Interpret Findings

Security - High Severity

Indicates that users' funds are at risk or that there is a high probability of exploitation.

Security - Medium Severity

No risk to the protocol or those who interact with it, however it should be highlighted nonetheless.

Logical - High Severity

Indicates that the errors puts users' funds at risk, or can result in significant functional failure in the code.

Logical - Medium Severity

Indicates some functional failure or discrepancy in the code.

Logical - Informational

Minor discrepancy between the intended functionality of the code and the implementation, which does not result in functional failure, or a recommendation to improve the logic.

Yellow Text

Indicates centralization of control and admin powers.

Disclaimer

The information in this deep logic audit report objectively describes the smart contracts being audited, and points out logical and mathematical errors, security risks and vulnerabilities, and optimization opportunities in the audited code. This deep logic audit does not ensure the correctness or authenticity of any software or dApp that interacts with or claims to interact with any smart contract.

This audit report does not constitute any advice whatsoever. You are solely responsible for conducting your own due diligence and consulting your financial advisor before making any investment decisions. While our deep logic audits raise the level of security, reliability, mathematical accuracy, and logical soundness of the smart contracts reviewed, they do not amount to any form of warranty or guarantee that the reviewed smart contracts are void of any weaknesses, vulnerabilities, or bugs. Prisma Shield and its founders, employees, owners, and associates are not liable for any damage or loss of funds. Please ensure trust in the team prior to investing as this deep logic audit does not guarantee the promised use of your funds.

You can find the full disclaimer, terms and conditions, and privacy policy on the prismashield.com website.

PRISMA SHIELD

Introducing Deep Logic Smart Contract Auditing to Web3

prismashield.com

prismashield@gmail.com

PrismaShield

PrismaShield