Bonsai3 IDO
Security Audit Report

PREPARED FOR:

Bonsai3 Crypto

ARCADIA CONTACT INFO

Email: audits@arcadiamgroup.com

Telegram: https://t.me/thearcadiagroup

Revision history

Date	Reason	Commit
01/08/2024	Initial Audit Scope	#0a8c2a6f542ed38b194a68d25368cb1379d96346
	Review Of Remediations	

Table of Contents

Executive Summary

1. Introduction and Audit Scope

Bonsai3 engaged Arcadia to perform a security audit of their main smart contracts version 2. Our review of their codebase occurred on the commit hash **#0a8c2a6f542ed38b194a68d25368cb1379d96346**

a. Review Team

Jihed Chalghaf - Security Researcher and Engineer

Joel Farris - Project Manager

b. Project Background

Bonsai3 offers a pioneering no-code toolkit designed by developers, aiming to be the WordPress equivalent for Web3. It empowers teams to deploy projects swiftly, affordably, and securely at any scale. The platform ensures community ownership and decentralized asset management, featuring unique tool sets for token creation, pre-sales, smart contract management, and market-making strategies.

c. Coverage

For this audit, we performed research, test coverage, investigation, and review of Bonsai3's main contract followed by issue reporting, along with mitigation and remediation instructions as outlined in this report. The following code repositories, files, and/or libraries are considered in scope for the review.

File	Lines	nLines	nSLOC	Comment Lines	Complex. Score
src/Bonsai3Launchpad.sol	541	428	375	20	221
src/FeeManager.sol	288	270	224	20	112
src/Libraries/CombinedEscrow.sol	179	156	120	19	101
src/Libraries/Bonsai3Sale.sol	84	54	47	6	15
src/TokenFactory.sol	41	35	29	1	47
src/Libraries/TokenDetails.sol	39	26	21	1	6
src/Interfaces/IFeeManager.sol	53	23	17	9	28

src/Libraries/Sale.sol	13	13	11	1	1
src/Libraries/Token.sol	12	12	10	1	1
src/Libraries/SelfDestruct.sol	13	10	7	1	1
src/Interfaces/ICombinedEscrow.sol	51	10	5	13	28
src/Interfaces/IBonsai3Launchpad.sol	83	6	3	5	44

2. Audit Summary

a. Audit Methodology

Arcadia completed this security review using various methods, primarily consisting of dynamic and static analysis. This process included a line-by-line analysis of the in-scope contracts, optimization analysis, analysis of key functionalities and limiters, and reference against intended functionality.

The followings are the steps we have performed while auditing the smart contracts:

- Investigating the project and its technical architecture overview through its documentation
- Understanding the overview of the smart contracts, the functions of the contracts, the inheritance, and how the contracts interface with each other thanks to the graph created by Solidity Visual Developer
- Manual smart contract audit:
    - Review the code to find any issue that could be exploited by known attacks listed by Consensys
    - Identifying which existing projects the smart contracts are built upon and what are the known vulnerabilities and remediations to the existing projects
    - Line-by-line manual review of the code to find any algorithmic and arithmetic related vulnerabilities compared to what should be done based on the project's documentation
    - Find any potential code that could be refactored to save gas
    - Run through the unit-tests and test-coverage if exists
- Static Analysis:
    - Scanning for vulnerabilities in the smart contracts using Static Code Analysis Software

    - Making a static analysis of the smart contracts using Slither
- Additional review: a follow-up review is done when the smart contracts have any new update. The follow-up is done by reviewing all changes compared to the audited commit revision and its impact to the existing source code and found issues.

b. Summary

There were **65** issues found, **7** of which were deemed to be ‘critical’, and **3** of which were rated as ‘high’. At the end of these issues were found throughout the review of a rapidly changing codebase and not a final static point in time.

Severity Rating	Number of Original Occurrences	Number of Remaining Occurrences
CRITICAL	7	7
HIGH	3	3
MEDIUM	16	16
LOW	14	14
INFORMATIONAL	15	15
GAS	10	10

Findings in Manual Audit

1. Missing input validation could cause **createNewToken** to pass without creating a token

Issue ID

BI3-1

Status

Unresolved

Risk Level

Severity: Critical, likelihood: High

Code Segment

```
function createNewToken(
        uint256 _id,
        uint256 _totalSupply,
        string calldata _tokenName,
        string calldata _tokenTicker,
        uint8 _template,
        address userWallet
    ) public payable override contractTurnedOn returns (address) {
        require(
            _totalSupply > 0,
            "You cannot send a negative supply of 0 or less!"
        );
        _handleFeePayment(_template);
        ChildToken storage newChildToken = tokenDetails[_id];
        require(newChildToken.id == 0, "Token has already been
created");
        newChildToken.id = _id;
        newChildToken.tokenTemplate = _template;
        newChildToken.owner = userWallet;
        newChildToken.totalSupply = _totalSupply;
        newChildToken.saleToken = _createToken(
            _tokenName,
            _tokenTicker,
            _totalSupply,
            _template,
            userWallet
```

```
        );
```

Description

During the creation of a new token, users specify **_template** to determine the token's type, which can be **Token** if **_template** equals one or **Seed** if _template equals two.

Otherwise, no tokens will be created, and the **TokenFactory::deployToken** function will return the new token's address as **address(0).**

If **_template** is greater than five, the transaction will be reverted during the fee payment process when calling **feeManager.getFeeAmount(_template)**, so there is no need to worry.

However, if **_template** <= 5, it could still be one of {0,3,4,5}, preventing the fee payment process from reverting because it accepts values in that range, thus disregarding the function's purpose.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Proof of concept

Consider adding this fuzz test into your existing tests at **test/Bonsai3Launchpad.t.sol.** It will pass when given a template <= 5

```
//@note fuzz test that accepts templates ⩽ 5 without reverting.
//The expected behavior is to only pass if template is in {1,2}
//however, a template in {0,3,4,5} does not cause the transaction to
revert.
//for template>5, the transaction will revert as expected.
function testFuzz_createNewToken(uint8 template) public {
    vm.assume(template ⩽ 5);
    vm.startPrank(user);
    IERC20(
        bonsai3.createNewToken{value: tokenTemplate2}( // using the
most expensive fee amount to avoid fee failure
            9999,
            9e23,
```

```
            "TEST",
            "TST",
            template,
            user
        )
    );
}
```

Recommendation

Consider validating **_template** at the start of the function:

```
require(_template == 1 || _template == 2, "createNewToken: invalid
template");
```

2. An off-by-one error may result in griefing attacks, potentially leading to locked funds

Issue ID

BI3-2

Status

Unresolved

Risk Level

Severity: Critical, likelihood: High

Description

The functions **createNewToken** and **createNewSale** do not require the **_id** input parameter to be greater than zero, resulting in bypassing the existing validations:

```
require(newChildToken.id == 0, "Token has already been created");
require(
            saleDetails[_id].id == 0,
            "createNewSale: Sale ID is incorrect"
        );
```

This means that a malicious user can either overwrite the token stored at the 0 id, preventing the original token owner at **id == 0** from listing their token for sale if they have not already done so.

Furthermore, a malicious user may choose to overwrite an existing sale with **id == 0**, locking the participants' funds because the sale's **escrowContract** is updated with each sale creation.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Proof of concept

–

Recommendation

Validate that the given **_id** is greater than zero in both functions.

3. The absence of sale token validation could result in the use of dangerous tokens

Issue ID

BI3-3

Status

Unresolved

Risk Level

Severity: Critical, likelihood: High

Code Segment

```
function createNewSale(
        uint256 _id,
        uint256 _totalSaleSupply,
```

```
        uint256 _presalePrice, // AKA to token price
        uint256 _softCap,
        uint256 _startTime,
        uint256 _endTime,
        address _saleToken,
        address _purchaseToken,
        address _userWallet
    ) public payable override contractTurnedOn {
        require(
            saleDetails[_id].id == 0,
            "createNewSale: Sale ID is incorrect"
        );
        require(
            _endTime > _startTime,
            "Your start time cannot be before your end time"
        );
        _handleFeePayment(4);
        _handleERCPayment(_saleToken, _totalSaleSupply);
        _setTokenState(_id, Token.TokenState.Insale);

        NewSale storage newUserSale = saleDetails[_id];
        newUserSale.id = _id;
        newUserSale.presalePrice = _presalePrice; // change to price *
token quantity.
        newUserSale.softCap = _softCap; //
        newUserSale.saleToken = _saleToken;
```

Description

The function **createNewSale** enables users to sell their created tokens via the launchpad. However, the function does not validate the **_saleToken** legitimacy.

A malicious user can start a sale using a dangerous token with a blacklisting feature. Then he can track the sale's progress, collect participants' addresses, and wait for it to reach the soft cap.

If there are any tokens left for sale, the malicious seller can take part in his own sale, allowing him to claim the tokens when the sale ends and make the most profit instead of waiting for the first participant to claim his tokens and blacklists the rest of participants.

Just before the sale's end, the seller blacklists all other participants, preventing them from using the **userClaimTokens** function because the sale token transfers will fail.

The seller finally calls **userClaimTokens**, which sends him the purchase tokens paid by the participants, while the sale tokens are stuck on the sale's **escrowContract** and cannot be withdrawn by **Bonsai3Launchpad** nor sent to their proper owners.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Proof of concept

–

Recommendation

Remove the **_saleToken** parameter and use **tokenDetails[_id].saleToken** instead.

4. Lack of **msg.value** validation may cause launchpad's funds to drain

Issue ID

BI3-4

Status

Unresolved

Risk Level

Severity: Critical, likelihood: High

Code Segment

```
function _handleFeePayment(uint8 key) internal {
        if (msg.value > 0) {
            require(msg.value > 0, "please add more eth to pay for
fees.");
        }
        require(
            feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(
                key, //todo last change should be made here @bufo
```

```
                msg.sender
            ),
            "please pay fee"
        );
    }
```

Description

The functions **createNewToken**, **createNewSale**, and **userClaimTokens** rely on the **_handleFeePayment** function, which does not ensure that the caller has sent enough ETH before calling **feeManager.chargeFeeByType**.

This could result in sending the required fee amount from the contract's funds, assuming the user has supplied sufficient ETH.

Additionally, fees can be in the form of ERC20 tokens. This means that the contract is always assuming that fees are in ETH and trying to send from its own funds, even when we do not need to send ETH.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Proof of concept

–

Recommendation

Consider changing to:

```
require(
            feeManager.chargeFeeByType{value: msg.value}(
                key, //todo last change should be made here @bufo
                msg.sender
            ),
            "please pay fee"
        );
```

This way, both fee currencies (native ETH and ERC20) will be managed by **feeManager.chargeFeeByType.**

5. Launchpad's flawed refund mechanism could result in locked funds

Issue ID

BI3-5

Status

Unresolved

Risk Level

Severity: Critical, likelihood: High

Code Segment

```
function userClaimTokens(uint256 _id) public override {
        ..
        ..
        } else if (saleLocalObj.saleState == Sale.SaleState.Refund) {
           saleLocalObj.escrowContract.withdraw(payable(msg.sender));
            emit UserClaimedRefund(
                _id,
                msg.sender,
                saleLocalObj.totalUserPurchases[msg.sender]
            );
        }
        ..
```

Description

In a sale where the state is set to **Refund**, any participant trying to claim their tokens will result in the contract refunding their purchase tokens, which can be either native ETH or ERC20 tokens. As shown in the implementation, the function only manages refunding purchase tokens in the form of native ETH. As a result, if the purchase token was an ERC20 token, the function will revert, resulting in the participants' purchase tokens being stuck in

the **Bonsai3Launchpad** contract and the sale owner's sale tokens being stuck in the **escrow** contract.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Proof of concept

- User creates a sale, accepting **USDC** as the purchase token.
- Multiple users take part in the sale without reaching the sale's **softCap**.
- Sale's **endTime** was reached.
- Any eligible participant calling **userClaimTokens** will result in the transaction reverting with no way to get their tokens back; the same applies for the sale owner's sale tokens.

Recommendation

Consider handling both cases when refunding purchase tokens.

```
if (saleLocalObj.purchaseToken == address(0)) {
  saleLocalObj.escrowContract.withdraw(payable(msg.sender));
} else {
  IERC20(saleLocalObj.purchaseToken).safeTransfer(
    msg.sender,
    saleLocalObj.totalUserPurchases[msg.sender]
  );
}
```

6. Absence of refund to the sale's owner will result in locked funds

Issue ID

BI3-6

Status

Unresolved

Risk Level

Severity: Critical, likelihood: High

Code Segment

At **Bonsai3Launchpad._endSale()**

```
if (saleLocalObj.totalTokensSold < saleLocalObj.softCap) {
  saleLocalObj.saleState = Sale.SaleState.Refund;
  saleLocalObj.escrowContract.enableRefunds();
}
```

Description

The first eligible participant to call **userClaimTokens** will pay for the sale state's update by entering **_endSale()**. In the case of a sale that has not reached the **softCap,** the sale state will be marked as **Refund** and the **escrowContract**'s state will also be updated to **Refunding.** The issue is that the contract is not sending the sale owner's tokens back, resulting in their tokens being stuck in both the **escrowContract** and the **Bonsai3Launchpad** contracts.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

```
if (saleLocalObj.totalTokensSold < saleLocalObj.softCap) {
  saleLocalObj.saleState = Sale.SaleState.Refund;
  saleLocalObj.escrowContract.enableRefunds();
  uint256 refundAmountAtEscrow =
saleLocalObj.escrowContract.getTotalERCBalance();
//1. refund the amount sent to the escrow through participateInSale
//calls
  saleLocalObj.escrowContract.withdrawAfterRefund();
//2. refund the remaining amount from this contract
  IERC20(saleLocalObj.saleToken).transfer(
    saleLocalObj.owner,
    saleLocalObj.totalSupply - refundAmountAtEscrow
  );
```

```
}
```

7. Erroneous payment check logic at **FeeManager** can lead to underflow

Issue ID

BI3-7

Status

Unresolved

Risk Level

Severity: Critical, likelihood: High

Code Segment

```
function _checkPayment(
        address payee,
        uint256 balanceBefore,
        uint256 requiredBalance,
        address currency
    ) internal returns (bool) {
        uint256 userBalance;
        if (currency == address(0)) {
            userBalance = payee.balance - balanceBefore;
        } else {
            userBalance = IERC20(currency).balanceOf(payee) -
balanceBefore;
        }
        emit PaymentSuccess(true, payee);
        return (userBalance ≥ requiredBalance);
    }
```

Description

For each fee-related function at **Bonsai3Launchpad (createNewSale, createNewToken, participateInSale)**, the contract calls **FeeManager. chargeFeeByType** which uses the above function at the end to confirm that the user has supplied sufficient funds. Nonetheless, the **userBalance** assignments are inversed since the function is subtracting

the pre-payment balance from the current balance, potentially leading to an underflow and causing unexpected behavior.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

```
if (currency == address(0)) {
            userBalance = balanceBefore - payee.balance;
        } else {
            userBalance = balanceBefore
-IERC20(currency).balanceOf(payee);
        }
```

8. Unhandled post-payment check at **FeeManager** could lead to users bypassing fee payments

Issue ID

BI3-8

Status

Unresolved

Risk Level

Severity: High, likelihood: Medium

Code Segment

```
function _handleTokenPayment(
        uint256 balanceBefore,
        IFeeManager.FeeTypes key,
        uint256 _purchaseAmount,
        address user
    ) internal {
        ..
        ..
        _checkPayment(user, balanceBefore, _purchaseAmount,
fees[key].currency);
```

```
    }
```

Description

The return value of the payment check function is not used, potentially allowing users to bypass fees payment, potentially draining the **Bonsai3Launchpad** ETH funds.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Validate the payment check return value by using it inside a **require** statement.

```
require(_checkPayment(user, balanceBefore, _purchaseAmount,
fees[key].currency), “Failed payment check.”);
```

9. Tokens with a fee-on-transfer mechanism will break the protocol

Issue ID

BI3-9

Status

Unresolved

Risk Level

Severity: High, likelihood: Medium

Code Segment

```
// at Bonsai3Launchpad
function _handleERCPayment(
        address _saleToken,
        uint256 amount
    ) internal returns (bool) {
        IERC20 saleToken = IERC20(_saleToken);
        uint256 balanceBefore = saleToken.balanceOf(address(this));
        saleToken.transferFrom(msg.sender, address(this), amount);
```

```
        return (saleToken.balanceOf(address(this)) ≥ balanceBefore +
amount);
    }
// at CombinedEscrow
function depositERC20(
        uint256 amount,
        address bonsaiHoldings,
        address payee
    ) external nonReentrant onlyOwner contractNotDestroyed {
        require(
            state() == State.Active,
            "ConditionalEscrow: depositERC20 Users can only deposit
tokens when sale is active."
        );
        require(amount > 0, "Amount must be greater than 0");
        require(
            _saleToken.transferFrom(bonsaiHoldings, address(this),
amount),
            "ConditionalEscrow: depositERC20 Transfer failed"
        );
        userErc20Balances[address(_saleToken)][payee] += amount;
        totalErcBalance += amount;
    }
// at CombinedEscrow
function withdrawERC20(
        address payee
    ) external nonReentrant onlyOwner contractNotDestroyed returns
(uint256) {
        require(
            state() == State.Closed,
            "ConditionalEscrow: withdrawERC20 Users can only deposit
tokens when sale is active."
        );
        uint256 amount =
userErc20Balances[address(_saleToken)][payee];
        require(amount > 0, "No funds to withdraw");
        userErc20Balances[address(_saleToken)][payee] = 0;
        require(
            _saleToken.transfer(payee, amount),
            "ConditionalEscrow: withdrawERC20 Transfer failed"
        );
        totalErcBalance -= amount;
        require(
            totalErcBalance ≥ 0,
            "ConditionalEscrow: ERC Balance insufficient"
        );
        return amount;
```

```
    }
// at CombinedEscrow
function withdrawAfterRefund()
        public
        nonReentrant
        onlyOwner
        contractNotDestroyed
    {
        require(
            state() == State.Refunding,
            "CombinedEscrow: saleOwner can only withdraw project
tokens when the sale is refunded"
        );
        require(
            _saleToken.transfer(address(beneficiary()),
totalErcBalance),
            "CombinedEscrow: withdrawAfterRefund transfer failed"
        );
    }
```

Description

During the creation and participation of sales, the function **_handleERCPayment** is utilized. The received amount will, however, be less than the transferred amount when transferring **fee-on-transfer** tokens. Sales involving these tokens will therefore have a lower supply than what is kept at **totalSupply**. Additionally, **totalErcBalance** and **userErc20Balances** mapping will contain improper amounts. Resulting in one hand, if the sale is set to an ending state, users may make more claims than they ought to, which would cause the final participant to claim to revert. On the other hand, if the sale is set to a refund state, the sale owner's refund call during the first participant's claim will revert since the actual escrow's balance is less than the registered **totalErcBalance**. Causing locked funds for both the sale's owner and the participants.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
bonsai3-smart-contracts/src/Libraries/CombinedEscrow.sol
```

Recommendation

Consider storing the actual sale supply received during sale creation and storing the correct amount purchased for each user during their participation and the correct balances on the escrow contract.

10. Usage of the wrong user balance may lead to invalid payment checks

Issue ID

BI3-10

Status

Unresolved

Risk Level

Severity: High, likelihood: High

Code Segment

```
function chargeFeeByType(
        // IFeeManager.FeeTypes key,
        uint8 key,
        address user
    ) public payable onlyBonsai3 noReentrant returns (bool) {
        FeeObject memory invoices = fees[catalouge[key]];
        invoices.amount = fees[catalouge[key]].amount;
        invoices.currency = fees[catalouge[key]].currency;
        FeesOccurred storage userPurchase = chargedFees[user];
        userPurchase.feeCodes.push(catalouge[key]);
        userPurchase.amount += fees[catalouge[key]].amount;
        userPurchase.currency = fees[catalouge[key]].currency;
        if (invoices.currency == address(0)) {
            require(
                msg.value ≥ fees[catalouge[key]].amount,
                "MSG.Value insufficient"
            );
            _handleTokenPayment(user.balance, catalouge[key],
msg.value, user);
            userPurchase.paid = true;
            return true;
        } else {
```

```
            _handleTokenPayment(
                IERC20(invoices.currency).balanceOf(user),
                catalouge[key],
                fees[catalouge[key]].amount,
                user
            );
            userPurchase.paid = true;
            return true;
        }
    }
```

Description

At **Bonsai3Launchpad._handleFeePayment**, the user was supposedly charged for the fee amount before the **FeeManager.chargeFeeByType** call. Which means that **user.balance** at **chargeFeeByType** represents the current user balance after paying fees, not the actual pre-payment balance that **_handleTokenPayment** is looking for. As a result, during the payment check, legitimate payments will be considered invalid. Possibly DOS every fee payment made with native ETH.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider changing to:

```
_handleTokenPayment(user.balance + msg.value, catalouge[key],
msg.value, user);
```

This change will only be valid if the fix from **BI3-4** is applied.

11. Missing validation may cause a user to take part in a sale without making the required payment and without any tokens to claim

Issue ID

BI3-11

Status

Unresolved

Risk Level

Severity: Medium, likelihood: Medium

Code Segment

```
function participateInSale(
        uint256 _id,
        uint256 _purchaseAmount
    )
        public
        payable
        override
        requiredSaleState(_id)
        dontHogTheSaleBro(_id, _purchaseAmount)
        returns (bool)
    {
        require(block.timestamp ⩽ saleDetails[_id].endTime, "Sale has
ended");
        require(saleDetails[_id].id == _id, "Sale ID is incorrect");
        _handleFeePayment(5);
        NewSale storage saleLocalObj = saleDetails[_id];
        if (saleLocalObj.saleState == Sale.SaleState.Pending) {
            _handleSaleState(saleLocalObj);
        }
        require(
            saleLocalObj.saleState == Sale.SaleState.Active,
            "participateInSale: sale pending"
        );
        uint256 msgValue = msg.value;
        msgValue -= feeManager.getFeeAmount(5); // Payment without fee
        _handleUserPurchase(saleLocalObj, _purchaseAmount);
        if (saleLocalObj.purchaseToken == address(0)) {
```

```
            _handleEthPayment(saleLocalObj);
        } else {
            _handleERCPayment(
                saleLocalObj.purchaseToken,
                saleLocalObj.participatedAmount
            );
        }
        saleLocalObj.totalUserPurchases[msg.sender] += saleLocalObj
            .participatedAmount; // add purchase token amount to total
        _handleTokenDeposit(
            saleLocalObj.escrowContract,
            _purchaseAmount,
            saleLocalObj.saleToken
        );

        participatedSales[msg.sender].push(saleLocalObj.id); // user
joined investor list;
        saleLocalObj.userList.push(msg.sender); // add user to sale
object list << these two indexes may be different but >>
        emit UserParticipates(
            msg.sender,
            _id,
            saleLocalObj.participatedAmount,
            saleLocalObj.totalUserTokens[msg.sender]
        );
        return true;
    }
```

Description

For each sale participant, the real purchase amount is **saleLocalObj.participatedAmount** which is being set at: `_handleUserPurchase(saleLocalObj, _purchaseAmount);`

However, this amount can be equal to zero if **presalePrice** equals **zero** and **_purchaseAmount** is less than **one hundred** because of the following:

```
function _handleUserPurchase(
        NewSale storage saleLocalObj,
        uint256 _purchaseAmount
    ) internal {
        // irrelevant code removed
        if (saleLocalObj.presalePrice != 0) {
            saleLocalObj.participatedAmount = _calculateTotalCost(
                saleLocalObj.id,
```

```
                _purchaseAmount
            );
        } else {
            saleLocalObj.participatedAmount = _calculateUserTokens( //
dutch auction price = 0
                saleLocalObj.id,
                _purchaseAmount
            );
        }
    }
```

```
function _calculateUserTokens(
        uint256 _id,
        uint256 _tokenQuantity
    ) internal view override returns (uint256) {
        return (((((_tokenQuantity * 1e18) /
saleDetails[_id].totalSupply) /
            100) * saleDetails[_id].totalSupply) / 1e18); // this is
the pricing function to modify for priced round.
    }
```

The above formula can be shortened to: `_tokenQuantity / 100`

In other words, users will not be able to buy less than 100 (in Wei) of any particular sale token. Rather, either the transaction will revert if **purchaseToken** is native ETH (caused by **Escrow::deposit()**) or they will pay the participation fees and become participants, but they won't receive any tokens at the conclusion of the sale.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Reevaluate the formula used at **_calculateUserTokens**.

Additionally, consider checking the **participatedAmount** value after its assignment:

```
_handleUserPurchase(saleLocalObj, _purchaseAmount);
require(saleLocalObj.participatedAmount > 0, "participateInSale:
unsufficient purchase amount");
```

12. A missing check of the **saleToken** state can result in the sale of an improper token

Issue ID

BI3-12

Status

Unresolved

Risk Level

Severity: Medium, likelihood: High

Code Segment

```
function createNewSale(
        uint256 _id,
        uint256 _totalSaleSupply,
        uint256 _presalePrice, // AKA to token price
        uint256 _softCap,
        uint256 _startTime,
        uint256 _endTime,
        address _saleToken,
        address _purchaseToken,
        address _userWallet
    ) public payable override contractTurnedOn {
        require(
            saleDetails[_id].id == 0,
            "createNewSale: Sale ID is incorrect"
        );
        require(
            _endTime > _startTime,
            "Your start time cannot be before your end time"
        );
        _handleFeePayment(4);
        _handleERCPayment(_saleToken, _totalSaleSupply);
```

Description

The **createNewSale** function allows users to offer their token for sale only once. The current implementation makes it impossible to determine whether the **_saleToken** has the

correct state. This increases the risk of users creating sales with different ids and the same **_saleToken**.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Proof of concept

–

Recommendation

Consider adding the following require statement:

```
require(tokenDetails[_id].tokenState == Token.TokenState.Created,
"createNewSale: invalid token state");
```

13. Insufficient input validation during sale creation can lead to stuck funds

Issue ID

BI3-13

Status

Unresolved

Risk Level

Severity: Medium, likelihood: Low

Code Segment

```
function createNewSale(
        uint256 _id,
        uint256 _totalSaleSupply,
        uint256 _presalePrice, // AKA to token price
        uint256 _softCap,
        uint256 _startTime,
        uint256 _endTime,
        address _saleToken,
```

```
        address _purchaseToken,
        address _userWallet
    ) public payable override contractTurnedOn {
        require(
            saleDetails[_id].id == 0,
            "createNewSale: Sale ID is incorrect"
        );
        require(
            _endTime > _startTime,
            "Your start time cannot be before your end time"
        );
        _handleFeePayment(4);
        // irrelevant code removed
}
```

Description

The user may inadvertently select a **_startTime** that is too far in the future when creating a sale. Which causes his sale tokens to become trapped in the **Bonsai3Launchpad** contract.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Consider validating that the **_startTime** field is not too far away in the future, for example it should be max 1 year in the future.

14. Launchpad sales are incompatible with custom tokens such as USDT

Issue ID

BI3-14

Status

Unresolved

Risk Level

Severity: Medium, likelihood: Medium

Code Segment

```
function _handleTokenDeposit(
        CombinedEscrow escrow,
        uint256 _purchaseAmount,
        address _saleToken
    ) internal {
        uint256 balanceBefore =
IERC20(_saleToken).balanceOf(address(escrow));
        IERC20(_saleToken).approve(address(escrow), _purchaseAmount);
        escrow.depositERC20(_purchaseAmount, address(this),
msg.sender);
        require(
            IERC20(_saleToken).balanceOf(address(escrow)) ⩾
                balanceBefore + _purchaseAmount,
            "Payment failure, final transaction balance is not
correct."
        );
    }
```

Description

Some ERC20 tokens (like **USDT**) do not work when changing the allowance from an existing non-zero allowance value to protect against front-running changes of approvals.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Set the allowance to zero before increasing the allowance and use **safeApprove** or **safeIncreaseAllowance** from Openzeppelin's **safeERC20**.

15. Fee-on-transfer tokens are not supported as sale tokens

Issue ID

BI3-15

Status

Unresolved

Risk Level

Severity: Medium, likelihood: Medium

Code Segment

```
function _handleTokenDeposit(
        CombinedEscrow escrow,
        uint256 _purchaseAmount,
        address _saleToken
    ) internal {
        uint256 balanceBefore =
IERC20(_saleToken).balanceOf(address(escrow));
        IERC20(_saleToken).approve(address(escrow), _purchaseAmount);
        escrow.depositERC20(_purchaseAmount, address(this),
msg.sender);
        require(
            IERC20(_saleToken).balanceOf(address(escrow)) ≥
                balanceBefore + _purchaseAmount,
            "Payment failure, final transaction balance is not
correct."
        );
    }
```

Description

The highlighted require statement will revert when transferring **fee-on-transfer** tokens because the actual received amount will be less.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

It is possible to update the **_handleTokenDeposit** function to check for the actual received amount by the escrow contract. To withdraw the correct amount during participants' claims, you should also think about updating **escrow.depositERC20** to store the actual received amount on the **userErc20Balances** mapping and the **totalErcBalance** state variable.

Or clearly document that only standard ERC20 tokens are supported. Which is unlikely since **Seed** also applies fees on transfer.

16. Not all ERC20 tokens **transfer/transferFrom** return a boolean

Issue ID

BI3-16

Status

Unresolved

Risk Level

Severity: Medium, likelihood: Medium

Code Segment

```
function depositERC20(
        uint256 amount,
        address bonsaiHoldings,
        address payee
    ) external nonReentrant onlyOwner contractNotDestroyed {
        require(
            state() == State.Active,
            "ConditionalEscrow: depositERC20 Users can only deposit
tokens when sale is active."
        );
        require(amount > 0, "Amount must be greater than 0");
        require(
            _saleToken.transferFrom(bonsaiHoldings, address(this),
amount),
```

```
            "ConditionalEscrow: depositERC20 Transfer failed"
        );
        userErc20Balances[address(_saleToken)][payee] += amount;
        totalErcBalance += amount;
    }
```

```
function _handleTokenPayment(
        uint256 balanceBefore,
        IFeeManager.FeeTypes key,
        uint256 _purchaseAmount,
        address user
    ) internal {
        if (fees[key].currency == address(0)) {
            require(
                user.balance ⩾ _purchaseAmount,
                "User does not have the balance"
            );
            // Using address(0) to represent native currency
            require(msg.value == _purchaseAmount, "Incorrect ETH
amount sent");
            // Transfer Ether to the recipient address
            // __vault.sendValue
            (bool sent, ) = payable(__vault).call{value:
_purchaseAmount}("");
            require(sent, "Failed to send Ether");
        } else {
            IERC20 saleToken = IERC20(fees[key].currency);
            require(
                saleToken.balanceOf(user) ⩾ _purchaseAmount,
                "User does not have the balance"
            );
            require(
                saleToken.transferFrom(user, __vault,
_purchaseAmount),
                "Token transfer failed"
            );
        }
        _checkPayment(user, balanceBefore, _purchaseAmount,
fees[key].currency);
    }
```

```
function withdrawERC20(
        address payee
    ) external nonReentrant onlyOwner contractNotDestroyed returns
(uint256) {
```

```
        require(
            state() == State.Closed,
            "ConditionalEscrow: withdrawERC20 Users can only deposit
tokens when sale is active."
        );
        uint256 amount =
userErc20Balances[address(_saleToken)][payee];
        require(amount > 0, "No funds to withdraw");
        userErc20Balances[address(_saleToken)][payee] = 0;
        require(
            _saleToken.transfer(payee, amount),
            "ConditionalEscrow: withdrawERC20 Transfer failed"
        );
        totalErcBalance -= amount;
        require(
            totalErcBalance >= 0,
            "ConditionalEscrow: ERC Balance insufficient"
        );
        return amount;
    }
function withdrawAfterRefund()
        public
        nonReentrant
        onlyOwner
        contractNotDestroyed
    {
        require(
            state() == State.Refunding,
            "CombinedEscrow: saleOwner can only withdraw project
tokens when the sale is refunded"
        );
        require(
            _saleToken.transfer(address(beneficiary()),
totalErcBalance),
            "CombinedEscrow: withdrawAfterRefund transfer failed"
        );
    }
```

Description

Some tokens (like USDT) do not implement the EIP20 standard correctly and their transfer/transferFrom function return void instead of a success boolean. Calling these functions with the correct EIP20 function signatures will revert.

Code location

```
bonsai3-smart-contracts/src/Libraries/CombinedEscrow.sol
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider using OpenZeppelin's **SafeERC20** versions with the **safeTransfer** and **safeTransferFrom** functions that manage the return value check as well as non-standard-compliant tokens.

```
// for example, replace the following require statement
require(
            _saleToken.transferFrom(bonsaiHoldings, address(this),
amount),
            "ConditionalEscrow: depositERC20 Transfer failed"
        );
// with
_saleToken.safeTransferFrom(bonsaiHoldings, address(this), amount);
```

17. Decrementing **totalTokensSold** after the sale's end causes an incorrect state

Issue ID

BI3-17

Status

Unresolved

Risk Level

Severity: Medium, likelihood: High

Code Segment

```
function userClaimTokens(uint256 _id) public override {
        NewSale storage saleLocalObj = saleDetails[_id];
        require(
            (saleDetails[_id].totalUserPurchases[msg.sender] > 0),
            "User did not participate"
        );
```

```
        if (saleLocalObj.saleState == Sale.SaleState.Active) {
            _endSale(saleLocalObj);
        }
        require(
            (saleLocalObj.saleState == Sale.SaleState.Ended ||
                saleLocalObj.saleState == Sale.SaleState.Refund),
            "Sale has not ended"
        );
        if (saleLocalObj.saleState == Sale.SaleState.Ended) {
            uint256 transferAmt =
saleLocalObj.escrowContract.withdrawERC20(
                msg.sender
            );
            saleLocalObj.totalTokensSold -= transferAmt;
// irrelevant code removed
}
```

Description

At the end of a sale where **X** tokens were sold, the sale's **totalTokensSold** variable will always be equal to 0 when it is not the case.

Additionally, it'll provide improper return values when calling **getSaleProgress()** and **getParticipatedAmount()**.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Consider adding a new field to the **Bonsai3Sale.NewSale** struct called for example **debtInTokens** and use it where fit it instead of **totalTokensSold**. Also, consider incrementing **totalTokensSold** each time a user claims a certain token amount.

```
// at Bonsai3Launchpad.userClaimTokens
saleLocalObj.totalTokensSold -= transferAmt;
saleLocalObj.totalTokensSold += transferAmt;
```

```
// at Bonsai3Launchpad.dontHogTheSaleBro
```

```
require(
            saleDetails[_id].totalTokensSold + _purchaseAmount ⩽
                saleDetails[_id].totalSupply,
            "Sale has reached hardcap"
        );
require(
            saleDetails[_id].debtInTokens + _purchaseAmount ⩽
                saleDetails[_id].totalSupply,
            "Sale has reached hardcap"
        );
```

```
// at Bonsai3Launchpad._handleUserPurchase
require(
  saleLocalObj.totalTokensSold + _purchaseAmount ⩽
  saleLocalObj.totalSupply,
  "You cannot buy more tokens than are for sale"
);
saleLocalObj.totalTokensSold += _purchaseAmount; // total tokens sold
for presale
}
require(
  saleLocalObj.debtInTokens + _purchaseAmount ⩽
  saleLocalObj.totalSupply,
  "You cannot buy more tokens than are for sale"
);
saleLocalObj.debtInTokens += _purchaseAmount; // total tokens sold for
presale
}
```

```
// at Bonsai3Launchpad._endSale
if (saleLocalObj.totalTokensSold < saleLocalObj.softCap)
if (saleLocalObj.debtInTokens < saleLocalObj.softCap)
```

18. Invalid check will result in sales not being marked as closed after conclusion

Issue ID

BI3-18

Status

Unresolved

Risk Level

Severity: Medium, likelihood: High

Code Segment

```
function userClaimTokens(uint256 _id) public override {
        NewSale storage saleLocalObj = saleDetails[_id];
        require(
            (saleDetails[_id].totalUserPurchases[msg.sender] > 0),
            "User did not participate"
        );
        if (saleLocalObj.saleState == Sale.SaleState.Active) {
            _endSale(saleLocalObj);
        }
        require(
            (saleLocalObj.saleState == Sale.SaleState.Ended ||
                saleLocalObj.saleState == Sale.SaleState.Refund),
            "Sale has not ended"
        );
        if (saleLocalObj.saleState == Sale.SaleState.Ended) {
            uint256 transferAmt =
saleLocalObj.escrowContract.withdrawERC20(
                msg.sender
            );
            saleLocalObj.totalTokensSold -= transferAmt; //
subtracting the total sale until closed. @auditor please tell me if
more efficient way to do this.
            emit UserClaimedTokens(_id, msg.sender, transferAmt);
        } else if (saleLocalObj.saleState == Sale.SaleState.Refund) {
            saleLocalObj.escrowContract.withdraw(payable(msg.sender));
            emit UserClaimedRefund(
                _id,
                msg.sender,
                saleLocalObj.totalUserPurchases[msg.sender]
            );
        }
        if (saleLocalObj.totalSupply == 0) {
            saleLocalObj.saleState = Sale.SaleState.Closed;
        }
    }
```

Description

The condition that is highlighted will never be true because **totalSupply** has never decreased. Which implies that the sales will never be marked as **Closed**.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Consider adding a new field **debtInTokens** to the **Bonsai3Sale.NewSale** struct, increment it each time a user takes part in a sale and increment the sale's **totalTokensSold** by **transferAmt** each time a user claims his tokens. Please check out the detailed recommendation on the related **BI3-17** issue.

Finally, replace the highlighted condition above with:

```
if(saleLocalObj.totalTokensSold == saleLocalObj.debtInTokens) {
  saleLocalObj.saleState = Sale.SaleState.Closed;
}
```

19. Faulty payment logic with a variety of tokens

Issue ID

BI3-19

Status

Unresolved

Risk Level

Severity: Medium, likelihood: High

Code Segment

```
struct FeesOccurred {
        IFeeManager.FeeTypes[] feeCodes;
        uint256 amount;
        bool paid;
        address currency;
```

```
    }
```

Description

The protocol accepts various tokens for paying fees, but the paid amount for each fee is aggregated inside the amount field without factoring token decimals (USDC: 6, WETH: 18, etc.). Additionally, currency reflects only the latest payment information, disregarding previous payments.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider storing all the currencies and the amount per currency.

Furthermore, the paid field is initially set to true when a user pays fees; after that, it remains set to true indefinitely and is reset to true each time a user pays a fee, which is a gas waste. Think again about the reasoning behind it.

```
struct FeesOccurred {
        IFeeManager.FeeTypes[] feeCodes;
        uint256[] amounts;
        bool paid;
        address[] currencies;
    }
```

20. Invalid check can result in fee payment failure even though user has provided enough ETH

Issue ID

BI3-20

Status

Unresolved

Risk Level

Severity: Medium, likelihood: Medium

Code Segment

```
function _handleTokenPayment(
        uint256 balanceBefore,
        IFeeManager.FeeTypes key,
        uint256 _purchaseAmount,
        address user
    ) internal {
        if (fees[key].currency == address(0)) {
            require(
                user.balance >= _purchaseAmount,
                "User does not have the balance"
            );
        // irrelevant code removed
```

Description

By determining whether **`user.balance`** exceeds **`_purchaseAmount`**, the **`_handleTokenPayment`** function attempts to determine whether the user has contributed enough ETH through the Bonsai3Launchpad contract. However, the balance used for the check was registered after the payment was made, so any funds that were sent to **`Bonsai3Launchpad`** prior to the **`FeeManager`** call had already been subtracted from the balance of the current user. Because they do not have a specific amount left, users may not pass the fees check consequently.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Proof of Concept

- Bob attempts to create a new Seed token with a current balance of 0.08 ETH (equivalent to 8e16 Wei)

- Bob initiates the **`Bonsai3Launchpad.createNewToken`** function, providing the required fee which currently equals 5e16 Wei and paying for the transaction gas fees. Bob's remaining balance now is a bit less than 3e16 Wei.
- During the token creation process, the fee payment is initiated. The highlighted require statement will look like this: **`require(~3e16 ≥ 5e16)`** which will revert with the given message **`"User does not have the balance".`**

Recommendation

Consider removing the highlighted require statement.

21. Unable to pay fees with excess received ETH

Issue ID

BI3-21

Status

Unresolved

Risk Level

Severity: Medium, likelihood: High

Code Segment

```
function _handleTokenPayment(
        uint256 balanceBefore,
        IFeeManager.FeeTypes key,
        uint256 _purchaseAmount,
        address user
    ) internal {
        if (fees[key].currency == address(0)) {
            require(
                user.balance ≥ _purchaseAmount,
                "User does not have the balance"
            );
            require(msg.value == _purchaseAmount, "Incorrect ETH
amount sent");
// irrelevant code removed
```

Description

The contract only accepts the received amount that matches _purchaseAmount when paying with ETH.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider changing to:

```
require(msg.value ≥ _purchaseAmount, "Incorrect ETH amount sent");
```

22. Absence of refund mechanism at **FeeManager**

Issue ID

BI3-22

Status

Unresolved

Risk Level

Severity: Medium, likelihood: High

Code Segment

```
function chargeFeeByType(
        // IFeeManager.FeeTypes key,
        uint8 key,
        address user
    ) public payable onlyBonsai3 noReentrant returns (bool) {
        FeeObject memory invoices = fees[catalouge[key]];
        invoices.amount = fees[catalouge[key]].amount;
        invoices.currency = fees[catalouge[key]].currency;
        FeesOccurred storage userPurchase = chargedFees[user];
        userPurchase.feeCodes.push(catalouge[key]);
        userPurchase.amount += fees[catalouge[key]].amount;
        userPurchase.currency = fees[catalouge[key]].currency;
        if (invoices.currency == address(0)) {
            require(
```

```
                msg.value >= fees[catalouge[key]].amount,
                "MSG.Value insufficient"
            );
            _handleTokenPayment(user.balance, catalouge[key],
msg.value, user);
            userPurchase.paid = true;
            return true;
        } else {
            _handleTokenPayment(
                IERC20(invoices.currency).balanceOf(user),
                catalouge[key],
                fees[catalouge[key]].amount,
                user
            );
            userPurchase.paid = true;
            return true;
        }
    }
```

Description

Users interacting with payable functions from the Bonsai3Launchpad contract can send a larger amount than the required fee, which will sent to the FeeManager contract, when applying the recommended mitigation from BI3-4. But the FeeManager contract does not give the user a refund for the extra ETH.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider implementing the following refund mechanism:

```
if (invoices.currency == address(0)) {
            require(
                msg.value >= fees[catalouge[key]].amount,
                "MSG.Value insufficient"
            );
            _handleTokenPayment(user.balance, catalouge[key],
fees[catalouge[key]].amount, user);
        if (msg.value > fees[catalouge[key]].amount) {
```

```
                (bool success, ) = payable(user).call{
                    value: msg.value - fees[catalouge[key]].amount
                }("");
                require(success, "ETH refund failed");
            }
            userPurchase.paid = true;
            return true;
        }
```

23. Usage of **transfer()** instead of **call()** when sending ETH

Issue ID

BI3-23

Status

Unresolved

Risk Level

Severity: Medium, likelihood: Medium

Code Segment

```
function beneficiaryWithdraw()
        public
        override
        nonReentrant
        onlyOwner
        contractNotDestroyed
    {
        require(
            state() == State.Closed,
            "CombinedEscrow: beneficiary can only withdraw while
closed"
        );
        beneficiary().transfer(((address(this).balance) * 95) / 100);
        vault.transfer(address(this).balance);
        require(address(this).balance == 0, "Balance issue");
    }
```

Description

This has some notable shortcomings when the recipient is a smart contract or a multisig wallet. The transfer will inevitably fail when the smart contract:

- does not implement a payable fallback function, or

- implements a payable fallback function which would incur more than 2300 gas units (applies to multisig wallets), or

- implements a payable fallback function incurring less than 2300 gas units but is called through a proxy that raises the call's gas usage above 2300.

Code location

```
bonsai3-smart-contracts/src/Libraries/CombinedEscrow.sol
```

Recommendation

Consider using **call()** with its returned Boolean checked in combination with **re-entrancy guard**.

24. Missing check could lead to taking part in a sale before its official start time

Issue ID

BI3-24

Status

Unresolved

Risk Level

Severity: Medium, likelihood: High

Code Segment

```
function participateInSale(
        uint256 _id,
        uint256 _purchaseAmount
    )
        public
        payable
        override
```

```
        requiredSaleState(_id)
        dontHogTheSaleBro(_id, _purchaseAmount)
        returns (bool)
    {
        require(block.timestamp ⩽ saleDetails[_id].endTime, "Sale has
ended");
        require(saleDetails[_id].id == _id, "Sale ID is incorrect");
        _handleFeePayment(5);
        // irrelevant code removed
```

Description

When taking part in a sale, the contract does not check whether the sale has started. As a result, users may participate in a sale that is supposed to begin at a later date.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Consider adding the following check:

```
function participateInSale(
        uint256 _id,
        uint256 _purchaseAmount
    )
        public
        payable
        override
        requiredSaleState(_id)
        dontHogTheSaleBro(_id, _purchaseAmount)
        returns (bool)
    {
        require(block.timestamp ⩾ saleDetails[_id].startTime, "Sale
has not started");
        require(block.timestamp ⩽ saleDetails[_id].endTime, "Sale has
ended");
```

25. Tautology expression

Issue ID

BI3-25

Status

Unresolved

Risk Level

Severity: Low, likelihood: High

Code Segment

```
function withdrawERC20(
        address payee
    ) external nonReentrant onlyOwner contractNotDestroyed returns
(uint256) {
        require(
            state() == State.Closed,
            "ConditionalEscrow: withdrawERC20 Users can only deposit
tokens when sale is active."
        );
        uint256 amount =
userErc20Balances[address(_saleToken)][payee];
        require(amount > 0, "No funds to withdraw");
        userErc20Balances[address(_saleToken)][payee] = 0;
        require(
            _saleToken.transfer(payee, amount),
            "ConditionalEscrow: withdrawERC20 Transfer failed"
        );
        totalErcBalance -= amount;
        require(
            totalErcBalance >= 0,
            "ConditionalEscrow: ERC Balance insufficient"
        );
        return amount;
    }
```

Description

totalErcBalance type is **uint256** so it will always be greater than or equal to 0.

Code location

```
bonsai3-smart-contracts/src/Libraries/CombinedEscrow.sol
```

Recommendation

Consider removing the highlighted tautology.

26. Missing post-transfer assignment can lead to an incorrect state

Issue ID

BI3-26

Status

Unresolved

Risk Level

Severity: Low, likelihood: High

Code Segment

```
function withdrawAfterRefund()
        public
        nonReentrant
        onlyOwner
        contractNotDestroyed
    {
        require(
            state() == State.Refunding,
            "CombinedEscrow: saleOwner can only withdraw project
tokens when the sale is refunded"
        );
        require(
            _saleToken.transfer(address(beneficiary()),
totalErcBalance),
            "CombinedEscrow: withdrawAfterRefund transfer failed"
        );
    }
```

Description

After sending **totalErcBalance** of the sale token to the beneficiary, the variable tracking the escrow's balance should be zeroed to maintain a proper state and avoid any unexpected behavior.

Code location

```
bonsai3-smart-contracts/src/Libraries/CombinedEscrow.sol
```

Recommendation

Make sure to set **totalErcBalance** to zero during the transfer process.

27. Invalid/misleading revert messages

Issue ID

BI3-27

Status

Unresolved

Risk Level

Severity: Low, likelihood: High

Code Segment

```
// at CombinedEscrow
function withdrawERC20(
        address payee
    ) external nonReentrant onlyOwner contractNotDestroyed returns
(uint256) {
        require(
            state() == State.Closed,
            "ConditionalEscrow: withdrawERC20 Users can only deposit
tokens when sale is active."
        );
uint256 amount = userErc20Balances[address(_saleToken)][payee];
        require(amount > 0, "No funds to withdraw");
```

```
// at FeeManager
```

```
modifier notPaused() {
        require(__paused ≠ true, "not paused.");
        _;
    }
```

```
// at Bonsai3Launchpad
function createNewToken(
        uint256 _id,
        uint256 _totalSupply,
        string calldata _tokenName,
        string calldata _tokenTicker,
        uint8 _template,
        address userWallet
    ) public payable override contractTurnedOn returns (address) {
        require(
            _totalSupply > 0,
            "You cannot send a negative supply of 0 or less!"
        );
```

Description

Throughout the code base, there are multiple invalid, or misleading revert messages that can cause users' confusion.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
bonsai3-smart-contracts/src/FeeManager.sol
bonsai3-smart-contracts/src/Libraries/CombinedEscrow.sol
```

Recommendation

Consider implementing the following changes:

```
-"ConditionalEscrow: withdrawERC20 Users can only deposit tokens when sale is active."
+"ConditionalEscrow: withdrawERC20 Users can only withdraw tokens when sale is closed."
-"No funds to withdraw"
+"ConditionalEscrow: withdrawERC20 insufficient payee balance"
-"not paused."
+"paused."
```

```
-"You cannot send a negative supply of 0 or less!"
+"Bonsai3Launchpad: createNewToken Insufficient supply"
```

28. Missing events on crucial functions and important state changes

Issue ID

BI3-28

Status

Unresolved

Risk Level

Severity: Low, likelihood: High

Description

It has been observed that important functionality is missing emitting events for some functions: **`CombinedEscrow .depositERC20`**, **`CombinedEscrow .withdrawERC20`**, **`Bonsai3Launchpad.setUp`**, **`Bonsai3Launchpad.updateFactory`** and **`Bonsai3Launchpad.updateFeeManager`**. These functions should emit events. Events are a method of informing the transaction initiator about the actions taken by the called function.

Code location

```
bonsai3-smart-contracts/src/Libraries/CombinedEscrow.sol
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Make sure to emit events in all essential functions.

29. Missing zero address checks could lead to redeploying FeeManager

Issue ID

BI3-29

Status

Unresolved

Risk Level

Severity: Low, likelihood: Low

Code Segment

```
constructor(address _bonsai_address, address payable _vault) {
        _grantRole(ADMIN_ROLE, msg.sender);
        _grantRole(BONSAI3, _bonsai_address);
        _setRoleAdmin(ADMIN_ROLE, ADMIN_ROLE);
        __vault = _vault;
```

Description

During the **FeeManager** deployment, the deployer can mistakenly provide a zero address for either **__vault** or **_bonsai_address** with no way of updating them after the deployment. This could result in redeploying the whole contract and loss of gas caused by the first deployment.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider adding the zero address checks for both state variables.

30. Hardcoding fees in USD at **FeeManager** constructor can be misleading

Issue ID

BI3-30

Status

Unresolved

Risk Level

Severity: Low, likelihood: High

Code Segment

```
constructor(address _bonsai_address, address payable _vault) {
        // irrelevant code removed
            0, //$0
        // irrelevant code removed
            880000000000000, //$2
        // irrelevant code removed
            50000000000000000, // $100
        // irrelevant code removed
            25000, // $5
        // irrelevant code removed
    }
```

Description

At **FeeManager**'s constructor, fees in Wei are commented with the equivalent values in **USD**.

Since ETH is not a stable coin, these comments can be misleading.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider removing the comments to avoid any confusion.

31. Misplaced **PaymentSuccess** event emission at FeeManager

Issue ID

BI3-31

Status

Unresolved

Risk Level

Severity: Low, likelihood: High

Code Segment

```
function _checkPayment(
        address payee,
        uint256 balanceBefore,
        uint256 requiredBalance,
        address currency
    ) internal returns (bool) {
        uint256 userBalance;
        if (currency == address(0)) {
            userBalance = payee.balance - balanceBefore;
        } else {
            userBalance = IERC20(currency).balanceOf(payee) -
balanceBefore;
        }
        emit PaymentSuccess(true, payee);
        return (userBalance ≥ requiredBalance);
    }
```

Description

The **PaymentSuccess** event is emitted at the **_checkPayment** function before checking the final condition which confirms the payment's validity. This could result in emitting a misleading successful payment event.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider changing the return statement of **_checkPayment** to void and replacing the return statement with a require statement: **require (userBalance ≥ requiredBalance).**

Then, emit the event after the **_checkPayment** call at **_handleTokenPayment**.

Alternatively, since the **_checkPayment** function is only used once, consider removing it and implementing the logic directly into **_handleTokenPayment** to save gas.

32. Presence of TODOs and comments implying unfinished code

Issue ID

BI3-32

Status

Unresolved

Risk Level

Severity: Low, likelihood: Low

Code Segment

```
// At FeeManager
//TODO set fee
//TODO Make payment token updateable
```

```
// At Bonsai3Launchpad
function _handleFeePayment(uint8 key) internal {
        if (msg.value > 0) {
            require(msg.value > 0, "please add more eth to pay for
fees.");
        }
```

```
        require(
            feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(
                key, //todo last change should be made here @bufo
                msg.sender
            ),
            "please pay fee"
        );
    }

// At Bonsai3Launchpad.participateInSale
saleLocalObj.userList.push(msg.sender); // add user to sale object
list << these two indexes may be different but >>
```

```
// At IFeeManager
// Maybe i can use this to change paymetn type
    enum PaymentCurrency {
        USDC,
        ETHEREUM,
        USDT,
        SEED
    }
```

Description

Open TODOs can point to architecture or programming issues that still need to be resolved.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
bonsai3-smart-contracts/src/Interfaces/IFeeManager.sol
```

Recommendation

Consider resolving the TODOs before deploying.

33. Lack of zero address validation when setting up **Bonsai3Launchpad** could result in contract redeployment

Issue ID

BI3-33

Status

Unresolved

Risk Level

Severity: Low, likelihood: Low

Code Segment

```
function setUp(
        IFactory _factory,
        IFeeManager _feemanager,
        address payable _vault
    ) public onlyRole(DEFAULT_ADMIN_ROLE) {
        require(_setup == false, "contract has already been setup");
        feeManager = _feemanager;
        _tokenFactory = _factory;
        __vault = _vault;
        _setup = true;
    }
```

Description

The **setUp** function is only called once for setting up the **bonsai3Launchpad** contract. As a result, it is important to validate the given addresses, essentially when the **__vault** state variable is not modifiable outside the **setUp** function, which could result in contract redeployment if things go wrong.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Consider implementing the required zero address checks.

34. Missing validation can lead to duplicate array items

Issue ID

BI3-34

Status

Unresolved

Risk Level

Severity: Low, likelihood: Medium

Code Segment

```
function participateInSale(
        uint256 _id,
        uint256 _purchaseAmount
    )
        public
        payable
        override
        requiredSaleState(_id)
        dontHogTheSaleBro(_id, _purchaseAmount)
        returns (bool)
    {
        // irrelevant code removed
        participatedSales[msg.sender].push(saleLocalObj.id); // user
joined investor list;
        saleLocalObj.userList.push(msg.sender); // add user to sale
object list << these two indexes may be different but >>
        emit UserParticipates(
            msg.sender,
            _id,
            saleLocalObj.participatedAmount,
            saleLocalObj.totalUserTokens[msg.sender]
        );
        return true;
    }
```

Description

If a user takes part X times in the same sale, the **`participatedSales[user]`** array will hold X duplicates of the sale's id. Additionally, the sale's **`userList`** array will hold X duplicates of the user's address.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Consider revising the logic behind using arrays.

35. Misuse of the "==" operator will result in an improper token state

Issue ID

BI3-35

Status

Unresolved

Risk Level

Severity: Low, likelihood: Medium

Code Segment

```
function _endSale(NewSale storage saleLocalObj) internal returns
(bool) {
        require(
            block.timestamp ≥ saleLocalObj.endTime,
            "Sale time has not elapsed"
        );
        if (saleLocalObj.totalTokensSold < saleLocalObj.softCap) {
            saleLocalObj.saleState = Sale.SaleState.Refund;
            saleLocalObj.escrowContract.enableRefunds();
        } else {
```

```
            saleLocalObj.saleState = Sale.SaleState.Ended;
            saleLocalObj.escrowContract.close();
            saleLocalObj.escrowContract.beneficiaryWithdraw();
            tokenDetails[saleLocalObj.id].tokenState ==
                Token.TokenState.Inmarket;
        }
        emit SaleIsOver(saleLocalObj.id, saleLocalObj.saleState);
        return true;
    }
```

Description

The use of the comparison operator instead of the assignment operator will leave the token in an incorrect state at the end of a sale.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Implement the following change:

```
            tokenDetails[saleLocalObj.id].tokenState =
                Token.TokenState.Inmarket;
```

36. Flawed self-destruction mechanism at CombinedEscrow

Issue ID

BI3-36

Status

Unresolved

Risk Level

Severity: Low, likelihood: High

Code Segment

```
function burnAfterReading() internal virtual override {
        _current = SelfDestruct.Active;
    }
modifier contractNotDestroyed() {
        require(
            _current == SelfDestruct.Inactive,
            "CombinedEscrow: Contract has been destroyed"
        );
        _;
    }
```

Description

The **burnAfterReading** function is never used in the **CombinedEscrow** contract. As a result, the **_current** state variable will always be equal to the default value which is **SelfDestruct.Inactive**, rendering the self-destruction mechanism irrelevant.

Code location

```
bonsai3-smart-contracts/src/Libraries/CombinedEscrow.sol
```

Recommendation

Consider revising the self-destruction mechanism.

37. Floating pragma version

Issue ID

BI3-37

Status

Unresolved

Risk Level

Severity: Low, likelihood: Medium

Description

The smart contracts in the project use floating version of Solidity **(^0.8.19)**. There is a list of known bugs in Solidity versions, many of them can affect smart contract functionality in unexpected way: https://github.com/ethereum/solidity/blob/develop/docs/bugs.json.

Furthermore, if compiled with **0.8.20** there may be unexpected reverts when deployed as some chains still do not support **PUSH0** (e.g., Optimism, BSC, etc.).

Code location

```
bonsai3-smart-contracts/src/*.sol
```

Recommendation

Consider using a static version.

38. Using an older version of **OpenZeppelin** libraries may be dangerous

Issue ID

BI3-38

Status

Unresolved

Risk Level

Severity: Low, likelihood: Medium

Code Segment

```
{
  "name": "openzeppelin-solidity",
  "description": "Secure Smart Contract library for Solidity",
  "version": "4.9.1",
..
}
```

Description

Currently the protocol is using version 4.9.1 of the **OpenZeppelin** contracts which are continuously updated to eliminate any bugs and vulnerabilities.

Code location

```
bonsai3-smart-contracts/lib/openzeppelin-contracts/package.json
bonsai3-smart-contracts/lib/openzeppelin-contracts-upgradeable/package.json
bonsai3-smart-contracts/lib/openzeppelin-solidity/package.json
```

Recommendation

Consider using the latest version **(5.0.1** so far**)**.

39. Unused code present in the codebase

Issue ID

BI3-39

Status

Unresolved

Risk Level

Severity: Informational

Code Segment

```
// At Bonsai3Sale
struct SaleData {
    uint256 totalSupply;
    uint256 presalePrice;
    uint256 softCap;
    uint256 startTime;
    uint256 endTime;
    uint256 totalTokensSold;
}
struct SaleAddress {
    address purchaseToken;
```

```
    address saleToken;
    address owner;
    address[] userList;
}
```

```
// At FeeManager
modifier feeMustExist(IFeeManager.FeeTypes feeId) {
    require(
        fees[IFeeManager.FeeTypes(feeId)].feeType ==
            IFeeManager.FeeTypes(feeId),
        "FeeId does not exist."
    );
    _;
}

modifier userMustExist(address user) {
    require(chargedFees[user].amount > 0, "No charges");
    _;
}
```

Description

Throughout the codebase, the highlighted structs and modifiers were never used.

Code location

```
bonsai3-smart-contracts/src/Libraries/Bonsai3Sale.sol
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

To improve the readability of the codebase and limit the potential attack surface, consider always removing unnecessary lines of code.

40. Inadequate naming can lead to confusion

Issue ID

BI3-40

Status

Unresolved

Risk Level

Severity: Informational

Description

Throughout the code base, there are multiple confusing, or misleading variable names.

At **`CombinedEscrow`** constructor: **`projectTreasury`** is not really the project's treasury address, it reflects the beneficiary state variable at the inherited **`Openzeppelin`**'s **`RefundEscrow`**. Additionally, **`getTotalERCBalance()`** and **`totalErcBalance`** do not reflect the nature of the ERC token, which is in this case an ERC20. Lastly, **`getERC20Balance`** can be renamed to **`getUserERC20Balance`** to enhance clarity.

At **`FeeManager`**, the **`catalogue`** mapping name is misspelled as **`catalouge`**. Additionally, at the **`_checkPayment`** function, the **`requiredBalance`** parameter is not really a balance, it represents the required amount that a user must pay. Added to that, in the same function, the **`userBalance`** variable is not the actual user's balance, it tends to refer to the user's paid amount.

At **`Bonsai3Launchpad`**, the **`_handleERCPayment`** function only handles ERC20 tokens, so it's better to indicate that on the name. In addition, this function was used on multiple occasions with both sale and purchase tokens which is contradictory with the **`_saleToken`** argument. Furthermore, **`getTokenDetails`** can be a little deceptive as it only returns a portion of the **`tokenDetails`** mapping. Aside from that, the return value of **`getTokenAddresses`** is not limited to addresses. It returns an **`uint8`** along with two addresses. Besides, **`getSalePurchaseToken`** and **`getTokenSaleAddress`** at **`Bonsai3Launchpad`** can be renamed to **`getPurchaseToken`** and **`getSaleToken`** to preserve consistency and improve clarity. Lastly, **`saleLocalObj`** at **`userClaimTokens()`** is not really

a local object since it is declared as **`storage`**, consider renaming to a more appropriate name such as **`sale`**.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
bonsai3-smart-contracts/src/Interfaces/IBonsai3Launchpad.sol
bonsai3-smart-contracts/src/FeeManager.sol
bonsai3-smart-contracts/src/Libraries/CombinedEscrow.sol
```

Recommendation

To favor explicitness and readability, we suggest renaming the variables and functions in question.

41. Incomplete comment

Issue ID

BI3-41

Status

Unresolved

Risk Level

Severity: Informational

Description

The following comment explaining **`depositERC20`** is incomplete: “depositERC20 is the function called after at the end of participate function. This will only be withdrawn to the”.

Code location

```
bonsai3-smart-contracts/src/Interfaces/ICombinedEscrow.sol
bonsai3-smart-contracts/src/Libraries/CombinedEscrow.sol
```

Recommendation

Consider finishing the comment to give a clear idea about the function.

42. Typo in comments

Issue ID

BI3-42

Status

Unresolved

Risk Level

Severity: Informational

Description

The following comments contain typos:

```
The owner of the reundescrow contract will be this contract which
holds the eth. combined can execute transactions on refund but only
through bonsai proxy.
```

```
Maybe i can use this to change paymetn type
```

Code location

```
bonsai3-smart-contracts/src/Libraries/CombinedEscrow.sol
bonsai3-smart-contracts/src/Interfaces/IFeeManager.sol
```

Recommendation

Consider changing to:

```
The owner of the RefundEscrow contract will be this contract which
holds the eth. Combined can execute transactions on refund but only
through bonsai proxy.
```

```
Maybe I can use this to change payment type
```

43. **IFeeManager** contains functions that are not defined on **FeeManager**

Issue ID

BI3-43

Status

Unresolved

Risk Level

Severity: Informational

Code Segment

```
function setSaleFeeRate(uint256 _saleFeeRate) external;

function buildCharge(FeeTypes key, address user) external;

function calculateSaleFee(
        uint256 saleAmount
    ) external view returns (uint256);

function payFees(address user) external;
```

Description

Functions that are not defined in the **FeeManager** contract are present in the **IFeeManager** interface. This leads to ambiguity, and contracts that communicate with the **FeeManager** via the interface could end up contacting functions that do not exist there.

Code location

```
bonsai3-smart-contracts/src/Interfaces/IFeeManager.sol
```

Recommendation

Consider removing those functions from the interface or implementing them on the **`FeeManager`** contract.

44. Redundant getter for **FeeManager.owner**

Issue ID

BI3-44

Status

Unresolved

Risk Level

Severity: Informational

Code Segment

```
// irrelevant code removed
address public owner;
function getOwner() external view returns (address) {
        return owner;
    }
```

Description

The compiler will generate a public getter for the **`owner`** property with the same name since it's a public contract variable.

Code location

```
bonsai3-smart-contracts/src/Interfaces/IFeeManager.sol
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider either:

- Removing **`getOwner()`** definition from the contract and renaming it to **`owner()`** at the interface.
- Changing the **`owner`** state variable visibility to **`private`**.

45. Function name does not represent the inner logic

Issue ID

BI3-45

Status

Unresolved

Risk Level

Severity: Informational

Code Segment

```
function _endSale(NewSale storage saleLocalObj) internal returns
(bool) {
        require(
            block.timestamp ≥ saleLocalObj.endTime,
            "Sale time has not elapsed"
        );
        if (saleLocalObj.totalTokensSold < saleLocalObj.softCap) {
            saleLocalObj.saleState = Sale.SaleState.Refund;
            saleLocalObj.escrowContract.enableRefunds();
        } else {
            saleLocalObj.saleState = Sale.SaleState.Ended;
            saleLocalObj.escrowContract.close();
            saleLocalObj.escrowContract.beneficiaryWithdraw();
            tokenDetails[saleLocalObj.id].tokenState ==
                Token.TokenState.Inmarket;
        }
        emit SaleIsOver(saleLocalObj.id, saleLocalObj.saleState);
        return true;
    }
```

Description

The function mentioned above has three different outcomes: it can either revert without changing the sale's state or mark its state to one of **`Ended`** / **`Refund`**. As a result, the present name can be misleading.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Consider renaming it to a more appropriate name such as **`_evaluateSale`** or **`_concludeSaleIfReady`**.

46. Improper visibility

Issue ID

BI3-46

Status

Unresolved

Risk Level

Severity: Informational

Description

The functions **`Bonsai3Launchpad.initialize`**, **`Bonsai3Launchpad.setUp`**, **`Bonsai3Launchpad.createNewToken`**, **`Bonsai3Launchpad.createNewSale`**, **`Bonsai3Launchpad.participateInSale`**, **`Bonsai3Launchpad.userClaimTokens`**, **`Bonsai3Launchpad.pause`**, **`Bonsai3Launchpad.unpause`**, **`Bonsai3Token.createNewToken`**, **`Bonsai3Sale.createNewSale`**, **`Bonsai3Sale.participateInSale`**, **`Bonsai3Sale.userClaimTokens`**,

FeeManager.haltOperations, **FeeManager.serThePatientMadeIt**, **FeeManager.chargeFeeByType**, **FeeManager.removeFee** and **TokenFactory.deployToken** visibilities are declared as **public**. However, these functions are intended to be called only from the outside. As a result, it is better to mark the visibility as **external**. It might also save gas since external functions read function arguments from **calldata** instead of having to allocate memory.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
bonsai3-smart-contracts/src/FeeManager.sol
bonsai3-smart-contracts/src/TokenFactory.sol
bonsai3-smart-contracts/src/Libraries/Bonsai3Sale.sol
bonsai3-smart-contracts/src/Libraries/Bonsai3Token.sol
```

Recommendation

Consider changing the visibility to **external** for the mentioned functions.

47. Uninitialized **FeeManager.owner** state variable

Issue ID

BI3-47

Status

Unresolved

Risk Level

Severity: Informational

Description

The **owner** state variable is never assigned throughout the **FeeManager** contract.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider assigning or removing the **`owner`** state variable.

48. FeeManager.__vault can be set as immutable

Issue ID

BI3-48

Status

Unresolved

Risk Level

Severity: Informational

Description

The **`__vault`** state variable is only assigned in the constructor. It can be declared as immutable to make it read-only, but only assignable in the constructor, reducing gas costs.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider marking the **`__vault`** state variable as **`immutable`**.

49. Using literals with too many digits reduces visibility

Issue ID

BI3-49

Status

Unresolved

Risk Level

Severity: Informational

Code Segment

```
constructor(address _bonsai_address, address payable _vault) {
        // irrelevant code removed

            880000000000000, //$2
            50000000000000000, // $100
    }
```

Description

At the **`FeeManager`**'s constructor, literals with too many digits were used when setting fees.

This causes ambiguity and reduces visibility.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider using scientific notations. For example, **`1000`** is equivalent to **`1e3`**.

50. Boolean equality

Issue ID

BI3-50

Status

Unresolved

Risk Level

Severity: Informational

Code Segment

```
// At FeeManager
modifier whenPaused() {
        require(__paused == true, "not paused.");
        _;
    }

modifier notPaused() {
        require(__paused != true, "not paused.");
        _;
    }
```

```
// At Bonsai3Launchpad
require(_setup == false, "contract has already been setup");
require(_setup == true, "Please Start Contract");
```

Description

Boolean equality was used on multiple occasions.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Change to the following:

```
// At FeeManager
modifier whenPaused() {
        require(__paused, "not paused.");
        _;
    }

modifier notPaused() {
        require(!__paused, "paused.");
        _;
    }
// At Bonsai3Launchpad
require(!_setup, "contract has already been setup");
require(_setup, "Please Start Contract");
```

51. Non-conformance to Solidity naming conventions

Issue ID

BI3-51

Status

Unresolved

Risk Level

Severity: Informational

Description

Throughout the code base, several non-conformances to Solidity naming convention were spotted. It is advisable to adhere to these naming conventions to improve clarity and avoid ambiguity. These issues are reported through Slither as follows:

```
INFO:Detectors:
Parameter
Bonsai3Launchpad.setUp(IFactory,IFeeManager,address)._factory
(src/Bonsai3Launchpad.sol#82) is not in mixedCase
Parameter
Bonsai3Launchpad.setUp(IFactory,IFeeManager,address)._feemanager
(src/Bonsai3Launchpad.sol#83) is not in mixedCase
Parameter Bonsai3Launchpad.setUp(IFactory,IFeeManager,address)._vault
(src/Bonsai3Launchpad.sol#84) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,ad
dress)._id (src/Bonsai3Launchpad.sol#117) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,ad
dress)._totalSupply (src/Bonsai3Launchpad.sol#118) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,ad
dress)._tokenName (src/Bonsai3Launchpad.sol#119) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,ad
dress)._tokenTicker (src/Bonsai3Launchpad.sol#120) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,ad
dress)._template (src/Bonsai3Launchpad.sol#121) is not in mixedCase
```

```
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256
,uint256,address,address,address)._id (src/Bonsai3Launchpad.sol#175)
is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256
,uint256,address,address,address)._totalSaleSupply
(src/Bonsai3Launchpad.sol#176) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256
,uint256,address,address,address)._presalePrice
(src/Bonsai3Launchpad.sol#177) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256
,uint256,address,address,address)._softCap
(src/Bonsai3Launchpad.sol#178) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256
,uint256,address,address,address)._startTime
(src/Bonsai3Launchpad.sol#179) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256
,uint256,address,address,address)._endTime
(src/Bonsai3Launchpad.sol#180) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256
,uint256,address,address,address)._saleToken
(src/Bonsai3Launchpad.sol#181) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256
,uint256,address,address,address)._purchaseToken
(src/Bonsai3Launchpad.sol#182) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256
,uint256,address,address,address)._userWallet
(src/Bonsai3Launchpad.sol#183) is not in mixedCase
Parameter Bonsai3Launchpad.participateInSale(uint256,uint256)._id
(src/Bonsai3Launchpad.sol#217) is not in mixedCase
Parameter
Bonsai3Launchpad.participateInSale(uint256,uint256)._purchaseAmount
(src/Bonsai3Launchpad.sol#218) is not in mixedCase
Parameter Bonsai3Launchpad.userClaimTokens(uint256)._id
(src/Bonsai3Launchpad.sol#376) is not in mixedCase
Parameter Bonsai3Launchpad.getListOfUsers(uint256)._id
(src/Bonsai3Launchpad.sol#429) is not in mixedCase
Parameter Bonsai3Launchpad.getSaleProgress(uint256)._id
(src/Bonsai3Launchpad.sol#434) is not in mixedCase
```

```
Parameter Bonsai3Launchpad.getParticipatedAmount(uint256)._id
(src/Bonsai3Launchpad.sol#447) is not in mixedCase
Parameter Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)._id
(src/Bonsai3Launchpad.sol#454) is not in mixedCase
Parameter
Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)._id
(src/Bonsai3Launchpad.sol#461) is not in mixedCase
Parameter Bonsai3Launchpad.getSalePurchaseToken(uint256)._id
(src/Bonsai3Launchpad.sol#467) is not in mixedCase
Parameter Bonsai3Launchpad.getTokenSaleAddress(uint256)._id
(src/Bonsai3Launchpad.sol#471) is not in mixedCase
Parameter Bonsai3Launchpad.getPreSalePrice(uint256)._id
(src/Bonsai3Launchpad.sol#475) is not in mixedCase
Parameter Bonsai3Launchpad.getSaleNumbers(uint256)._id
(src/Bonsai3Launchpad.sol#480) is not in mixedCase
Parameter Bonsai3Launchpad.getTokenDetails(uint256)._id
(src/Bonsai3Launchpad.sol#504) is not in mixedCase
Parameter Bonsai3Launchpad.getTokenAddresses(uint256)._id
(src/Bonsai3Launchpad.sol#514) is not in mixedCase
Parameter Bonsai3Launchpad.updateFactory(address)._newFactory
(src/Bonsai3Launchpad.sol#531) is not in mixedCase
Parameter Bonsai3Launchpad.updateFeeManager(address)._feeManager
(src/Bonsai3Launchpad.sol#537) is not in mixedCase
Variable Bonsai3Launchpad.__deployer (src/Bonsai3Launchpad.sol#69) is
not in mixedCase
Variable Bonsai3Launchpad.__vault (src/Bonsai3Launchpad.sol#70) is not
in mixedCase
Variable Bonsai3Launchpad._tokenFactory (src/Bonsai3Launchpad.sol#73)
is not in mixedCase
Variable Bonsai3Launchpad._setup (src/Bonsai3Launchpad.sol#74) is not
in mixedCase
Variable FeeManager.__paused (src/FeeManager.sol#11) is not in
mixedCase
Variable FeeManager.__vault (src/FeeManager.sol#14) is not in
mixedCase
Parameter
Factory.deployToken(string,string,uint256,uint256,address)._name
(src/TokenFactory.sol#15) is not in mixedCase
Parameter
Factory.deployToken(string,string,uint256,uint256,address)._ticker
(src/TokenFactory.sol#16) is not in mixedCase
Parameter
Factory.deployToken(string,string,uint256,uint256,address)._supply
(src/TokenFactory.sol#17) is not in mixedCase
Parameter
Factory.deployToken(string,string,uint256,uint256,address)._template
(src/TokenFactory.sol#18) is not in mixedCase
```

```
Parameter
Factory.deployToken(string,string,uint256,uint256,address)._from
(src/TokenFactory.sol#19) is not in mixedCase
```

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
bonsai3-smart-contracts/src/TokenFactory.sol
```

Recommendation

Follow the Solidity naming convention.

52. Lacking comments and **NatSpec** documentation

Issue ID

BI3-52

Status

Unresolved

Risk Level

Severity: Informational

Description

The documentation's lacking makes it hard to understand the code and to review that the implemented functionality matches the intended behavior.

Code location

```
bonsai3-smart-contracts/src/*.sol
```

Recommendation

Consider adding clear comments when needed and use **NatSpec** comments for documenting every public interface.

53. Order of functions is not respected

Status

Unresolved

Risk Level

Severity: Informational

Description

Ordering helps readers identify which functions they can call and to find the constructor and fallback definitions easier.

Code location

```
bonsai3-smart-contracts/src/*.sol
```

Recommendation

Consider applying the order of functions for each contract.

54. Extra space at revert message

Issue ID

BI3-54

Status

Unresolved

Risk Level

Severity: Gas

Code Segment

```
"CombinedEscrow:  Please send a non zero amount"
```

Description

Every character count when using revert messages, removing unwanted characters saves a bit of gas. Additionally, there is a presence of a grammatical error that should be fixed.

Code location

```
bonsai3-smart-contracts/src/Libraries/CombinedEscrow.sol
```

Recommendation

Change to the following:

```
"CombinedEscrow: Please send a non-zero amount"
```

55. Redundant pre-transfer check

Issue ID

BI3-55

Status

Unresolved

Risk Level

Severity: Gas

Code Segment

```
function _handleTokenPayment(
        uint256 balanceBefore,
        IFeeManager.FeeTypes key,
        uint256 _purchaseAmount,
        address user
    ) internal {
        // irrelevant code removed
        } else {
            IERC20 saleToken = IERC20(fees[key].currency);
            require(
                saleToken.balanceOf(user) ≥ _purchaseAmount,
                "User does not have the balance"
            );
```

```
            require(
                saleToken.transferFrom(user, __vault,
_purchaseAmount),
                "Token transfer failed"
            );
        }
        _checkPayment(user, balanceBefore, _purchaseAmount,
fees[key].currency);
    }
```

Description

Since the user's balance will be checked during the transfer, it is unnecessary to check it before starting the transfer. Furthermore, even if the user has the necessary funds to transfer, the transaction may still fail if the **`FeeManager`** contract does not have sufficient allowance.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider removing the highlighted require statement.

56. Duplicate assignments

Issue ID

BI3-56

Status

Unresolved

Risk Level

Severity: Gas

Code Segment

```
function chargeFeeByType(
        // IFeeManager.FeeTypes key,
        uint8 key,
        address user
    ) public payable onlyBonsai3 noReentrant returns (bool) {
        FeeObject memory invoices = fees[catalouge[key]];
        invoices.amount = fees[catalouge[key]].amount;
        invoices.currency = fees[catalouge[key]].currency;
```

Description

Duplicate assignments are made for the **amount** and **currency** fields after the **invoices** variable has been assigned.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider removing the duplicate assignments.

57. Reading from storage after assigning a local memory variable

Issue ID

BI3-57

Status

Unresolved

Risk Level

Severity: Gas

Code Segment

```
function chargeFeeByType(
```

```
        // IFeeManager.FeeTypes key,
        uint8 key,
        address user
    ) public payable onlyBonsai3 noReentrant returns (bool) {
        FeeObject memory invoices = fees[catalouge[key]];
        invoices.amount = fees[catalouge[key]].amount;
        invoices.currency = fees[catalouge[key]].currency;
        FeesOccurred storage userPurchase = chargedFees[user];
        userPurchase.feeCodes.push(catalouge[key]);
        userPurchase.amount += fees[catalouge[key]].amount;
        userPurchase.currency = fees[catalouge[key]].currency;
        if (invoices.currency == address(0)) {
            require(
                msg.value >= fees[catalouge[key]].amount,
                "MSG.Value insufficient"
            );
            _handleTokenPayment(user.balance, catalouge[key],
msg.value, user);
            userPurchase.paid = true;
            return true;
        } else {
            _handleTokenPayment(
                IERC20(invoices.currency).balanceOf(user),
                catalouge[key],
                fees[catalouge[key]].amount,
                user
            );
            userPurchase.paid = true;
            return true;
        }
    }
```

Description

After assigning the **`invoices`** memory variable, it is more gas-efficient to use it instead of reading multiple times from the contract's storage.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Consider replacing the highlighted occurrences with the **`invoices`** variable.

58. Unused local variable assignment

Issue ID

BI3-58

Status

Unresolved

Risk Level

Severity: Gas

Code Segment

```
function participateInSale(
        uint256 _id,
        uint256 _purchaseAmount
    )
        public
        payable
        override
        requiredSaleState(_id)
        dontHogTheSaleBro(_id, _purchaseAmount)
        returns (bool)
    {
        require(block.timestamp ⩽ saleDetails[_id].endTime, "Sale has
ended");
        require(saleDetails[_id].id == _id, "Sale ID is incorrect");
        _handleFeePayment(5);
        NewSale storage saleLocalObj = saleDetails[_id];
        if (saleLocalObj.saleState == Sale.SaleState.Pending) {
            _handleSaleState(saleLocalObj);
        }
        require(
            saleLocalObj.saleState == Sale.SaleState.Active,
            "participateInSale: sale pending"
        );
        uint256 msgValue = msg.value;
        msgValue -= feeManager.getFeeAmount(5); // Payment without fee
        _handleUserPurchase(saleLocalObj, _purchaseAmount);
```

```
        if (saleLocalObj.purchaseToken == address(0)) {
            _handleEthPayment(saleLocalObj);
        } else {
            _handleERCPayment(
                saleLocalObj.purchaseToken,
                saleLocalObj.participatedAmount
            );
        }
        saleLocalObj.totalUserPurchases[msg.sender] += saleLocalObj
            .participatedAmount; // add purchase token amount to total
        _handleTokenDeposit(
            saleLocalObj.escrowContract,
            _purchaseAmount,
            saleLocalObj.saleToken
        );

        participatedSales[msg.sender].push(saleLocalObj.id); // user
joined investor list;
        saleLocalObj.userList.push(msg.sender); // add user to sale
object list << these two indexes may be different but >>
        emit UserParticipates(
            msg.sender,
            _id,
            saleLocalObj.participatedAmount,
            saleLocalObj.totalUserTokens[msg.sender]
        );
        return true;
    }
```

Description

After assigning and decrementing the **`msgValue`** local variable, it has remained unutilized.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Consider removing the highlighted lines or reassess the logic behind them.

59. Meaningless if clause

Issue ID

BI3-59

Status

Unresolved

Risk Level

Severity: Gas

Code Segment

```
function _handleFeePayment(uint8 key) internal {
        if (msg.value > 0) {
            require(msg.value > 0, "please add more eth to pay for
fees.");
        }
```

Description

The if clause mentioned above adds nothing but gas consumption that is not necessary.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Consider removing the highlighted if clause.

60. Redundant check

Issue ID

BI3-60

Status

Unresolved

Risk Level

Severity: Gas

Code Segment

```
function _handleSaleState(NewSale storage saleLocalObj) internal {
        // First user to participate in sale will spend the gas to
write state object
        if (saleLocalObj.saleState == Sale.SaleState.Pending) {
            require(
                saleLocalObj.startTime <= block.timestamp,
                "Sale has not started yet"
            );
            saleLocalObj.saleState = Sale.SaleState.Active; //
required sale to be active
        }
    }
function participateInSale(
        uint256 _id,
        uint256 _purchaseAmount
    )
        public
        payable
        override
        requiredSaleState(_id)
        dontHogTheSaleBro(_id, _purchaseAmount)
        returns (bool)
    {
        require(block.timestamp <= saleDetails[_id].endTime, "Sale has
ended");
        require(saleDetails[_id].id == _id, "Sale ID is incorrect");
        _handleFeePayment(5);
        NewSale storage saleLocalObj = saleDetails[_id];
        if (saleLocalObj.saleState == Sale.SaleState.Pending) {
            _handleSaleState(saleLocalObj);
        }
```

Description

The if clause at **_handleSaleState** is redundant since the same condition is being validated at **participateInSale** before calling **_handleSaleState** as shown.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Consider **either** leaving **participateInSale** unchanged and change **_handleSaleState** to:

```
function _handleSaleState(NewSale storage saleLocalObj) internal {
        // First user to participate in sale will spend the gas to
write state object
        require(
            saleLocalObj.startTime ⩽ block.timestamp,
            "Sale has not started yet"
        );
        saleLocalObj.saleState = Sale.SaleState.Active; // required
sale to be active
    }
```

Or leave **_handleSaleState** unchanged and change **participateInSale** as follows:

```
function participateInSale(
        uint256 _id,
        uint256 _purchaseAmount
    )
        public
        payable
        override
        requiredSaleState(_id)
        dontHogTheSaleBro(_id, _purchaseAmount)
        returns (bool)
    {
        require(block.timestamp ⩽ saleDetails[_id].endTime, "Sale has
ended");
        require(saleDetails[_id].id == _id, "Sale ID is incorrect");
        _handleFeePayment(5);
        NewSale storage saleLocalObj = saleDetails[_id];
        _handleSaleState(saleLocalObj);
```

61. Redundant validation

Issue ID

BI3-61

Status

Unresolved

Risk Level

Severity: Gas

Code Segment

```
function _handleUserPurchase(
        NewSale storage saleLocalObj,
        uint256 _purchaseAmount
    ) internal {
        require(
            saleLocalObj.totalTokensSold + _purchaseAmount ⩽
                saleLocalObj.totalSupply,
            "You cannot buy more tokens than are for sale"
        );
modifier dontHogTheSaleBro(uint256 _id, uint256 _purchaseAmount) {
        require(
            saleDetails[_id].totalTokensSold + _purchaseAmount ⩽
                saleDetails[_id].totalSupply,
            "Sale has reached hardcap"
        );
        _;
    }
```

Description

The highlighted validation was already made when calling **participateInSale** through the **dontHogTheSaleBro** modifier prior to calling **_handleUserPurchase.**

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Consider removing the highlighted require statement from the **_handleUserPurchase** function.

62. Duplicate validation

Issue ID

BI3-62

Status

Unresolved

Risk Level

Severity: Gas

Code Segment

```
function _calculateTotalCost(
        uint256 _id,
        uint256 _tokenQuantity
    ) internal view override returns (uint256) {
        require(saleDetails[_id].presalePrice > 0);
        return (((_tokenQuantity) * saleDetails[_id].presalePrice));
    }
function _handleUserPurchase(
        // irrelevant code removed

        if (saleLocalObj.presalePrice != 0) {
            saleLocalObj.participatedAmount = _calculateTotalCost(
                saleLocalObj.id,
                _purchaseAmount
            );
        }
```

Description

The highlighted validation was already made prior to the function call at **_handleUserPurchase** since **saleLocalObj** was initially assigned as: **NewSale storage saleLocalObj = saleDetails[_id];**

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Consider removing the highlighted require statement from the **`_calculateTotalCost`** function.

63. Use custom errors instead of require statements

Issue ID

BI3-63

Status

Unresolved

Risk Level

Severity: Gas

Description

Custom errors are available from solidity version **`0.8.4`**. Instead of using error strings, to reduce deployment and runtime cost, you should use custom errors.

Code location

```
bonsai3-smart-contracts/src/*.sol
```

Recommendation

Consider replacing the require statements with **`if (something) revert CustomError()`** type of checks.

Automated Audit

Static Analysis with Slither

We run a static analysis against the source code using Slither, which is a Solidity static analysis framework written in Python 3. Slither runs a suite of vulnerability detectors, prints visual information about contract details. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses.

The following shows the results found by the static analysis by Slither. We reviewed the results, and, except the Reentrancy issues that are identified as above, all the other issues found by Slither are false positives.

```
INFO:Detectors:
FeeManager._handleTokenPayment(uint256,IFeeManager.FeeTypes,uint256,address)
(src/FeeManager.sol#164-193) uses arbitrary from in transferFrom:
require(bool,string)(saleToken.transferFrom(user,__vault,_purchaseAmount),Token transfer
failed) (src/FeeManager.sol#187-190)
CombinedEscrow.depositERC20(uint256,address,address)
(src/Libraries/CombinedEscrow.sol#111-127) uses arbitrary from in transferFrom:
require(bool,string)(_saleToken.transferFrom(bonsaiHoldings,address(this),amount),Conditio
nalEscrow: depositERC20 Transfer failed) (src/Libraries/CombinedEscrow.sol#121-124)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfr
om
INFO:Detectors:
Bonsai3Launchpad._handleFeePayment(uint8) (src/Bonsai3Launchpad.sol#336-347) sends eth to
arbitrary user
        Dangerous calls:
        - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
Disperse.disperseEther(address[],uint256[]) (src/Disperse.sol#15-21) sends eth to
arbitrary user
        Dangerous calls:
        - address(recipients[i]).transfer(values[i]) (src/Disperse.sol#17)
        - address(msg.sender).transfer(balance) (src/Disperse.sol#20)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to
-arbitrary-destinations
INFO:Detectors:
Reentrancy in FeeManager.chargeFeeByType(uint8,address) (src/FeeManager.sol#200-230):
        External calls:
        - _handleTokenPayment(user.balance,catalouge[key],msg.value,user)
(src/FeeManager.sol#217)
                - (sent) = address(__vault).call{value: _purchaseAmount}()
(src/FeeManager.sol#179)
```

```
                -
require(bool,string)(saleToken.transferFrom(user,__vault,_purchaseAmount),Token transfer
failed) (src/FeeManager.sol#187-190)
        External calls sending eth:
        - _handleTokenPayment(user.balance,catalouge[key],msg.value,user)
(src/FeeManager.sol#217)
                - (sent) = address(__vault).call{value: _purchaseAmount}()
(src/FeeManager.sol#179)
        State variables written after the call(s):
        - userPurchase.paid = true (src/FeeManager.sol#218)
        FeeManager.chargedFees (src/FeeManager.sol#54) can be used in cross function
reentrancies:
        - FeeManager.chargeFeeByType(uint8,address) (src/FeeManager.sol#200-230)
        - FeeManager.chargedFees (src/FeeManager.sol#54)
Reentrancy in FeeManager.chargeFeeByType(uint8,address) (src/FeeManager.sol#200-230):
        External calls:
        -
_handleTokenPayment(IERC20(invoices.currency).balanceOf(user),catalouge[key],fees[catalou
e[key]].amount,user) (src/FeeManager.sol#221-226)
                - (sent) = address(__vault).call{value: _purchaseAmount}()
(src/FeeManager.sol#179)
                -
require(bool,string)(saleToken.transferFrom(user,__vault,_purchaseAmount),Token transfer
failed) (src/FeeManager.sol#187-190)
        External calls sending eth:
        -
_handleTokenPayment(IERC20(invoices.currency).balanceOf(user),catalouge[key],fees[catalou
e[key]].amount,user) (src/FeeManager.sol#221-226)
                - (sent) = address(__vault).call{value: _purchaseAmount}()
(src/FeeManager.sol#179)
        State variables written after the call(s):
        - userPurchase.paid = true (src/FeeManager.sol#227)
        FeeManager.chargedFees (src/FeeManager.sol#54) can be used in cross function
reentrancies:
        - FeeManager.chargeFeeByType(uint8,address) (src/FeeManager.sol#200-230)
        - FeeManager.chargedFees (src/FeeManager.sol#54)
Reentrancy in
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214):
        External calls:
        - _handleFeePayment(4) (src/Bonsai3Launchpad.sol#193)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        - _handleERCPayment(_saleToken,_totalSaleSupply) (src/Bonsai3Launchpad.sol#194)
                - saleToken.transferFrom(msg.sender,address(this),amount)
(src/Bonsai3Launchpad.sol#317)
        External calls sending eth:
        - _handleFeePayment(4) (src/Bonsai3Launchpad.sol#193)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        State variables written after the call(s):
        - newUserSale.id = _id (src/Bonsai3Launchpad.sol#198)
        Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
```

```
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
        - newUserSale.presalePrice = _presalePrice (src/Bonsai3Launchpad.sol#199)
       Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
        - newUserSale.softCap = _softCap (src/Bonsai3Launchpad.sol#200)
       Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
```

```
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
        - newUserSale.saleToken = _saleToken (src/Bonsai3Launchpad.sol#201)
       Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
        - newUserSale.startTime = _startTime (src/Bonsai3Launchpad.sol#202)
       Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
```

```
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
        - newUserSale.endTime = _endTime (src/Bonsai3Launchpad.sol#203)
        Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
        - newUserSale.totalSupply = _totalSaleSupply (src/Bonsai3Launchpad.sol#204)
        Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
```

```
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
        - newUserSale.owner = _userWallet (src/Bonsai3Launchpad.sol#205)
        Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
        - newUserSale.purchaseToken = _purchaseToken (src/Bonsai3Launchpad.sol#206)
        Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
```

```
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
        - newUserSale.saleState = Sale.SaleState.Pending (src/Bonsai3Launchpad.sol#207)
        Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
        - newUserSale.escrowContract = new
CombinedEscrow(address(__vault),address(_userWallet),IERC20(_saleToken))
(src/Bonsai3Launchpad.sol#208-212)
        Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
```

```
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
Reentrancy in Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,address)
(src/Bonsai3Launchpad.sol#116-146):
        External calls:
        - _handleFeePayment(_template) (src/Bonsai3Launchpad.sol#128)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        - newChildToken.saleToken =
_createToken(_tokenName,_tokenTicker,_totalSupply,_template,userWallet)
(src/Bonsai3Launchpad.sol#135-141)
                -
(_tokenFactory.deployToken(_tokenName,_tokenTicker,_totalSupply,_template,_from))
(src/Bonsai3Launchpad.sol#163-171)
        External calls sending eth:
        - _handleFeePayment(_template) (src/Bonsai3Launchpad.sol#128)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        State variables written after the call(s):
        - _setTokenState(_id,Token.TokenState.Created) (src/Bonsai3Launchpad.sol#142)
                - tokenDetails[_id].tokenState = _state (src/Bonsai3Launchpad.sol#152)
        Bonsai3Launchpad.tokenDetails (src/Bonsai3Launchpad.sol#77) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._endSale(Bonsai3Sale.NewSale)
(src/Bonsai3Launchpad.sol#409-426)
        - Bonsai3Launchpad._setTokenState(uint256,Token.TokenState)
(src/Bonsai3Launchpad.sol#148-153)
        - Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,address)
(src/Bonsai3Launchpad.sol#116-146)
        - Bonsai3Launchpad.getTokenAddresses(uint256) (src/Bonsai3Launchpad.sol#513-521)
        - Bonsai3Launchpad.getTokenDetails(uint256) (src/Bonsai3Launchpad.sol#503-511)
        - Bonsai3Launchpad.tokenDetails (src/Bonsai3Launchpad.sol#77)
Reentrancy in Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266):
        External calls:
        - _handleFeePayment(5) (src/Bonsai3Launchpad.sol#229)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
```

```
        - _handleEthPayment(saleLocalObj) (src/Bonsai3Launchpad.sol#242)
                - saleLocalObj.escrowContract.deposit{value:
saleLocalObj.participatedAmount}(address(msg.sender)) (src/Bonsai3Launchpad.sol#298-300)
        - _handleERCPayment(saleLocalObj.purchaseToken,saleLocalObj.participatedAmount)
(src/Bonsai3Launchpad.sol#244-247)
                - saleToken.transferFrom(msg.sender,address(this),amount)
(src/Bonsai3Launchpad.sol#317)
        External calls sending eth:
        - _handleFeePayment(5) (src/Bonsai3Launchpad.sol#229)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        - _handleEthPayment(saleLocalObj) (src/Bonsai3Launchpad.sol#242)
                - saleLocalObj.escrowContract.deposit{value:
saleLocalObj.participatedAmount}(address(msg.sender)) (src/Bonsai3Launchpad.sol#298-300)
        State variables written after the call(s):
        - saleLocalObj.totalUserPurchases[msg.sender] += saleLocalObj.participatedAmount
(src/Bonsai3Launchpad.sol#249-250)
        Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
Reentrancy in Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266):
        External calls:
        - _handleFeePayment(5) (src/Bonsai3Launchpad.sol#229)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        - _handleEthPayment(saleLocalObj) (src/Bonsai3Launchpad.sol#242)
                - saleLocalObj.escrowContract.deposit{value:
saleLocalObj.participatedAmount}(address(msg.sender)) (src/Bonsai3Launchpad.sol#298-300)
        - _handleERCPayment(saleLocalObj.purchaseToken,saleLocalObj.participatedAmount)
(src/Bonsai3Launchpad.sol#244-247)
                - saleToken.transferFrom(msg.sender,address(this),amount)
(src/Bonsai3Launchpad.sol#317)
```

```
        -
_handleTokenDeposit(saleLocalObj.escrowContract,_purchaseAmount,saleLocalObj.saleToken)
(src/Bonsai3Launchpad.sol#251-255)
                - IERC20(_saleToken).approve(address(escrow),_purchaseAmount)
(src/Bonsai3Launchpad.sol#327)
                - escrow.depositERC20(_purchaseAmount,address(this),msg.sender)
(src/Bonsai3Launchpad.sol#328)
        External calls sending eth:
        - _handleFeePayment(5) (src/Bonsai3Launchpad.sol#229)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        - _handleEthPayment(saleLocalObj) (src/Bonsai3Launchpad.sol#242)
                - saleLocalObj.escrowContract.deposit{value:
saleLocalObj.participatedAmount}(address(msg.sender)) (src/Bonsai3Launchpad.sol#298-300)
        State variables written after the call(s):
        - saleLocalObj.userList.push(msg.sender) (src/Bonsai3Launchpad.sol#258)
        Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
Bonsai3Launchpad._handleERCPayment(address,uint256) (src/Bonsai3Launchpad.sol#311-319)
ignores return value by saleToken.transferFrom(msg.sender,address(this),amount)
(src/Bonsai3Launchpad.sol#317)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
FeeManager.owner (src/FeeManager.sol#13) is never initialized. It is used in:
        - FeeManager.getOwner() (src/FeeManager.sol#281-283)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variable
s
INFO:Detectors:
```

```
Bonsai3Launchpad._calculateUserTokens(uint256,uint256) (src/Bonsai3Launchpad.sol#360-366)
performs a multiplication on the result of a division:
        - (((((_tokenQuantity * 1e18) / saleDetails[_id].totalSupply) / 100) *
saleDetails[_id].totalSupply) / 1e18) (src/Bonsai3Launchpad.sol#364-365)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
CombinedEscrow.beneficiaryWithdraw() (src/Libraries/CombinedEscrow.sol#76-90) uses a
dangerous strict equality:
        - require(bool,string)(address(this).balance == 0,Balance issue)
(src/Libraries/CombinedEscrow.sol#89)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in Bonsai3Launchpad.userClaimTokens(uint256)
(src/Bonsai3Launchpad.sol#376-407):
        External calls:
        - _endSale(saleLocalObj) (src/Bonsai3Launchpad.sol#383)
                - saleLocalObj.escrowContract.enableRefunds()
(src/Bonsai3Launchpad.sol#416)
                - saleLocalObj.escrowContract.close() (src/Bonsai3Launchpad.sol#419)
                - saleLocalObj.escrowContract.beneficiaryWithdraw()
(src/Bonsai3Launchpad.sol#420)
        - transferAmt = saleLocalObj.escrowContract.withdrawERC20(msg.sender)
(src/Bonsai3Launchpad.sol#391-393)
        State variables written after the call(s):
        - saleLocalObj.totalTokensSold -= transferAmt (src/Bonsai3Launchpad.sol#394)
        Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
Reentrancy in Bonsai3Launchpad.userClaimTokens(uint256)
(src/Bonsai3Launchpad.sol#376-407):
        External calls:
        - _endSale(saleLocalObj) (src/Bonsai3Launchpad.sol#383)
                - saleLocalObj.escrowContract.enableRefunds()
(src/Bonsai3Launchpad.sol#416)
```

```
                - saleLocalObj.escrowContract.close() (src/Bonsai3Launchpad.sol#419)
                - saleLocalObj.escrowContract.beneficiaryWithdraw()
(src/Bonsai3Launchpad.sol#420)
        - transferAmt = saleLocalObj.escrowContract.withdrawERC20(msg.sender)
(src/Bonsai3Launchpad.sol#391-393)
        - saleLocalObj.escrowContract.withdraw(address(msg.sender))
(src/Bonsai3Launchpad.sol#397)
        State variables written after the call(s):
        - saleLocalObj.saleState = Sale.SaleState.Closed (src/Bonsai3Launchpad.sol#405)
        Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76) can be used in cross
function reentrancies:
        - Bonsai3Launchpad._calculateTotalCost(uint256,uint256)
(src/Bonsai3Launchpad.sol#368-374)
        - Bonsai3Launchpad._calculateUserTokens(uint256,uint256)
(src/Bonsai3Launchpad.sol#360-366)
        -
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214)
        - Bonsai3Launchpad.dontHogTheSaleBro(uint256,uint256)
(src/Bonsai3Launchpad.sol#107-114)
        - Bonsai3Launchpad.getListOfUsers(uint256) (src/Bonsai3Launchpad.sol#428-432)
        - Bonsai3Launchpad.getParticipatedAmount(uint256)
(src/Bonsai3Launchpad.sol#446-451)
        - Bonsai3Launchpad.getPreSalePrice(uint256) (src/Bonsai3Launchpad.sol#475-477)
        - Bonsai3Launchpad.getSaleNumbers(uint256) (src/Bonsai3Launchpad.sol#479-501)
        - Bonsai3Launchpad.getSaleProgress(uint256) (src/Bonsai3Launchpad.sol#434-438)
        - Bonsai3Launchpad.getSalePurchaseToken(uint256)
(src/Bonsai3Launchpad.sol#467-469)
        - Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)
(src/Bonsai3Launchpad.sol#460-465)
        - Bonsai3Launchpad.getTokenSaleAddress(uint256) (src/Bonsai3Launchpad.sol#471-473)
        - Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)
(src/Bonsai3Launchpad.sol#453-458)
        - Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266)
        - Bonsai3Launchpad.requiredSaleState(uint256) (src/Bonsai3Launchpad.sol#98-105)
        - Bonsai3Launchpad.saleDetails (src/Bonsai3Launchpad.sol#76)
        - Bonsai3Launchpad.userClaimTokens(uint256) (src/Bonsai3Launchpad.sol#376-407)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
CombinedEscrow.withdrawERC20(address) (src/Libraries/CombinedEscrow.sol#130-150) contains
a tautology or contradiction:
        - require(bool,string)(totalErcBalance >= 0,ConditionalEscrow: ERC Balance
insufficient) (src/Libraries/CombinedEscrow.sol#145-148)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction
INFO:Detectors:
Bonsai3Launchpad._handleTokenDeposit(CombinedEscrow,uint256,address)
(src/Bonsai3Launchpad.sol#321-334) ignores return value by
IERC20(_saleToken).approve(address(escrow),_purchaseAmount) (src/Bonsai3Launchpad.sol#327)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
CombinedEscrow.depositERC20(uint256,address,address)
(src/Libraries/CombinedEscrow.sol#111-127) should emit an event for:
        - totalErcBalance += amount (src/Libraries/CombinedEscrow.sol#126)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
Bonsai3Launchpad.setUp(IFactory,IFeeManager,address)._vault (src/Bonsai3Launchpad.sol#84)
lacks a zero-check on :
                - __vault = _vault (src/Bonsai3Launchpad.sol#89)
```

```
FeeManager.constructor(address,address)._vault (src/FeeManager.sol#56) lacks a zero-check
on :
                - __vault = _vault (src/FeeManager.sol#60)
CombinedEscrow.constructor(address,address,IERC20).__vault
(src/Libraries/CombinedEscrow.sol#31) lacks a zero-check on :
                - vault = __vault (src/Libraries/CombinedEscrow.sol#35)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validat
ion
INFO:Detectors:
Disperse.disperseEther(address[],uint256[]) (src/Disperse.sol#15-21) has external calls
inside a loop: address(recipients[i]).transfer(values[i]) (src/Disperse.sol#17)
Disperse.disperseToken(IERC20,address[],uint256[]) (src/Disperse.sol#23-30) has external
calls inside a loop: require(bool)(token.transfer(recipients[j],values[j]))
(src/Disperse.sol#29)
Disperse.disperseTokenSimple(IERC20,address[],uint256[]) (src/Disperse.sol#32-35) has
external calls inside a loop:
require(bool)(token.transferFrom(msg.sender,recipients[i],values[i]))
(src/Disperse.sol#34)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214):
        External calls:
        - _handleFeePayment(4) (src/Bonsai3Launchpad.sol#193)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        - _handleERCPayment(_saleToken,_totalSaleSupply) (src/Bonsai3Launchpad.sol#194)
                - saleToken.transferFrom(msg.sender,address(this),amount)
(src/Bonsai3Launchpad.sol#317)
        External calls sending eth:
        - _handleFeePayment(4) (src/Bonsai3Launchpad.sol#193)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        State variables written after the call(s):
        - _setTokenState(_id,Token.TokenState.Insale) (src/Bonsai3Launchpad.sol#195)
                - tokenDetails[_id].tokenState = _state (src/Bonsai3Launchpad.sol#152)
Reentrancy in Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,address)
(src/Bonsai3Launchpad.sol#116-146):
        External calls:
        - _handleFeePayment(_template) (src/Bonsai3Launchpad.sol#128)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        State variables written after the call(s):
        - newChildToken.id = _id (src/Bonsai3Launchpad.sol#131)
        - newChildToken.tokenTemplate = _template (src/Bonsai3Launchpad.sol#132)
        - newChildToken.owner = userWallet (src/Bonsai3Launchpad.sol#133)
        - newChildToken.totalSupply = _totalSupply (src/Bonsai3Launchpad.sol#134)
Reentrancy in Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,address)
(src/Bonsai3Launchpad.sol#116-146):
        External calls:
        - _handleFeePayment(_template) (src/Bonsai3Launchpad.sol#128)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
```

```
        - newChildToken.saleToken =
_createToken(_tokenName,_tokenTicker,_totalSupply,_template,userWallet)
(src/Bonsai3Launchpad.sol#135-141)
                -
(_tokenFactory.deployToken(_tokenName,_tokenTicker,_totalSupply,_template,_from))
(src/Bonsai3Launchpad.sol#163-171)
        External calls sending eth:
        - _handleFeePayment(_template) (src/Bonsai3Launchpad.sol#128)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        State variables written after the call(s):
        - userCreatedTokens[userWallet].push(_id) (src/Bonsai3Launchpad.sol#143)
Reentrancy in Factory.deployToken(string,string,uint256,uint256,address)
(src/TokenFactory.sol#14-36):
        External calls:
        - seed = new Seed(_name,_ticker,_from,_supply) (src/TokenFactory.sol#28)
        - seed.transferOwnership(_from) (src/TokenFactory.sol#29)
        State variables written after the call(s):
        - tokenCount += 1 (src/TokenFactory.sol#31)
        - tokens.push(address(seed)) (src/TokenFactory.sol#30)
Reentrancy in CombinedEscrow.depositERC20(uint256,address,address)
(src/Libraries/CombinedEscrow.sol#111-127):
        External calls:
        -
require(bool,string)(_saleToken.transferFrom(bonsaiHoldings,address(this),amount),Conditio
nalEscrow: depositERC20 Transfer failed) (src/Libraries/CombinedEscrow.sol#121-124)
        State variables written after the call(s):
        - totalErcBalance += amount (src/Libraries/CombinedEscrow.sol#126)
        - userErc20Balances[address(_saleToken)][payee] += amount
(src/Libraries/CombinedEscrow.sol#125)
Reentrancy in Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266):
        External calls:
        - _handleFeePayment(5) (src/Bonsai3Launchpad.sol#229)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        - _handleEthPayment(saleLocalObj) (src/Bonsai3Launchpad.sol#242)
                - saleLocalObj.escrowContract.deposit{value:
saleLocalObj.participatedAmount}(address(msg.sender)) (src/Bonsai3Launchpad.sol#298-300)
        - _handleERCPayment(saleLocalObj.purchaseToken,saleLocalObj.participatedAmount)
(src/Bonsai3Launchpad.sol#244-247)
                - saleToken.transferFrom(msg.sender,address(this),amount)
(src/Bonsai3Launchpad.sol#317)
        -
_handleTokenDeposit(saleLocalObj.escrowContract,_purchaseAmount,saleLocalObj.saleToken)
(src/Bonsai3Launchpad.sol#251-255)
                - IERC20(_saleToken).approve(address(escrow),_purchaseAmount)
(src/Bonsai3Launchpad.sol#327)
                - escrow.depositERC20(_purchaseAmount,address(this),msg.sender)
(src/Bonsai3Launchpad.sol#328)
        External calls sending eth:
        - _handleFeePayment(5) (src/Bonsai3Launchpad.sol#229)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        - _handleEthPayment(saleLocalObj) (src/Bonsai3Launchpad.sol#242)
                - saleLocalObj.escrowContract.deposit{value:
saleLocalObj.participatedAmount}(address(msg.sender)) (src/Bonsai3Launchpad.sol#298-300)
        State variables written after the call(s):
```

```
        - participatedSales[msg.sender].push(saleLocalObj.id)
(src/Bonsai3Launchpad.sol#257)
Reentrancy in CombinedEscrow.withdrawERC20(address)
(src/Libraries/CombinedEscrow.sol#130-150):
        External calls:
        - require(bool,string)(_saleToken.transfer(payee,amount),ConditionalEscrow:
withdrawERC20 Transfer failed) (src/Libraries/CombinedEscrow.sol#140-143)
        State variables written after the call(s):
        - totalErcBalance -= amount (src/Libraries/CombinedEscrow.sol#144)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Bonsai3Launchpad._endSale(Bonsai3Sale.NewSale)
(src/Bonsai3Launchpad.sol#409-426):
        External calls:
        - saleLocalObj.escrowContract.enableRefunds() (src/Bonsai3Launchpad.sol#416)
        - saleLocalObj.escrowContract.close() (src/Bonsai3Launchpad.sol#419)
        - saleLocalObj.escrowContract.beneficiaryWithdraw() (src/Bonsai3Launchpad.sol#420)
        Event emitted after the call(s):
        - SaleIsOver(saleLocalObj.id,saleLocalObj.saleState)
(src/Bonsai3Launchpad.sol#424)
Reentrancy in FeeManager._handleTokenPayment(uint256,IFeeManager.FeeTypes,uint256,address)
(src/FeeManager.sol#164-193):
        External calls:
        - (sent) = address(__vault).call{value: _purchaseAmount}()
(src/FeeManager.sol#179)
        - require(bool,string)(saleToken.transferFrom(user,__vault,_purchaseAmount),Token
transfer failed) (src/FeeManager.sol#187-190)
        External calls sending eth:
        - (sent) = address(__vault).call{value: _purchaseAmount}()
(src/FeeManager.sol#179)
        Event emitted after the call(s):
        - PaymentSuccess(true,payee) (src/FeeManager.sol#244)
                - _checkPayment(user,balanceBefore,_purchaseAmount,fees[key].currency)
(src/FeeManager.sol#192)
Reentrancy in
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address) (src/Bonsai3Launchpad.sol#174-214):
        External calls:
        - _handleFeePayment(4) (src/Bonsai3Launchpad.sol#193)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        - _handleERCPayment(_saleToken,_totalSaleSupply) (src/Bonsai3Launchpad.sol#194)
                - saleToken.transferFrom(msg.sender,address(this),amount)
(src/Bonsai3Launchpad.sol#317)
        External calls sending eth:
        - _handleFeePayment(4) (src/Bonsai3Launchpad.sol#193)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        Event emitted after the call(s):
        - CreateSale(_id,_endTime - _startTime) (src/Bonsai3Launchpad.sol#213)
Reentrancy in Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,address)
(src/Bonsai3Launchpad.sol#116-146):
        External calls:
        - _handleFeePayment(_template) (src/Bonsai3Launchpad.sol#128)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
```

```
        - newChildToken.saleToken =
_createToken(_tokenName,_tokenTicker,_totalSupply,_template,userWallet)
(src/Bonsai3Launchpad.sol#135-141)
                -
(_tokenFactory.deployToken(_tokenName,_tokenTicker,_totalSupply,_template,_from))
(src/Bonsai3Launchpad.sol#163-171)
        External calls sending eth:
        - _handleFeePayment(_template) (src/Bonsai3Launchpad.sol#128)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        Event emitted after the call(s):
        - CreatedToken(_tokenName,_tokenTicker,_totalSupply)
(src/Bonsai3Launchpad.sol#162)
                - newChildToken.saleToken =
_createToken(_tokenName,_tokenTicker,_totalSupply,_template,userWallet)
(src/Bonsai3Launchpad.sol#135-141)
        - NewToken(_id,userWallet,newChildToken.saleToken) (src/Bonsai3Launchpad.sol#144)
Reentrancy in Factory.deployToken(string,string,uint256,uint256,address)
(src/TokenFactory.sol#14-36):
        External calls:
        - seed = new Seed(_name,_ticker,_from,_supply) (src/TokenFactory.sol#28)
        - seed.transferOwnership(_from) (src/TokenFactory.sol#29)
        Event emitted after the call(s):
        - TokenDeployed(address(seed)) (src/TokenFactory.sol#32)
Reentrancy in Bonsai3Launchpad.participateInSale(uint256,uint256)
(src/Bonsai3Launchpad.sol#216-266):
        External calls:
        - _handleFeePayment(5) (src/Bonsai3Launchpad.sol#229)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        - _handleEthPayment(saleLocalObj) (src/Bonsai3Launchpad.sol#242)
                - saleLocalObj.escrowContract.deposit{value:
saleLocalObj.participatedAmount}(address(msg.sender)) (src/Bonsai3Launchpad.sol#298-300)
        - _handleERCPayment(saleLocalObj.purchaseToken,saleLocalObj.participatedAmount)
(src/Bonsai3Launchpad.sol#244-247)
                - saleToken.transferFrom(msg.sender,address(this),amount)
(src/Bonsai3Launchpad.sol#317)
        -
_handleTokenDeposit(saleLocalObj.escrowContract,_purchaseAmount,saleLocalObj.saleToken)
(src/Bonsai3Launchpad.sol#251-255)
                - IERC20(_saleToken).approve(address(escrow),_purchaseAmount)
(src/Bonsai3Launchpad.sol#327)
                - escrow.depositERC20(_purchaseAmount,address(this),msg.sender)
(src/Bonsai3Launchpad.sol#328)
        External calls sending eth:
        - _handleFeePayment(5) (src/Bonsai3Launchpad.sol#229)
                - require(bool,string)(feeManager.chargeFeeByType{value:
feeManager.getFeeAmount(key)}(key,msg.sender),please pay fee)
(src/Bonsai3Launchpad.sol#340-346)
        - _handleEthPayment(saleLocalObj) (src/Bonsai3Launchpad.sol#242)
                - saleLocalObj.escrowContract.deposit{value:
saleLocalObj.participatedAmount}(address(msg.sender)) (src/Bonsai3Launchpad.sol#298-300)
        Event emitted after the call(s):
        -
UserParticipates(msg.sender,_id,saleLocalObj.participatedAmount,saleLocalObj.totalUserToke
ns[msg.sender]) (src/Bonsai3Launchpad.sol#259-264)
Reentrancy in Bonsai3Launchpad.userClaimTokens(uint256)
(src/Bonsai3Launchpad.sol#376-407):
        External calls:
        - _endSale(saleLocalObj) (src/Bonsai3Launchpad.sol#383)
```

```
                - saleLocalObj.escrowContract.enableRefunds()
(src/Bonsai3Launchpad.sol#416)
                - saleLocalObj.escrowContract.close() (src/Bonsai3Launchpad.sol#419)
                - saleLocalObj.escrowContract.beneficiaryWithdraw()
(src/Bonsai3Launchpad.sol#420)
        - transferAmt = saleLocalObj.escrowContract.withdrawERC20(msg.sender)
(src/Bonsai3Launchpad.sol#391-393)
        Event emitted after the call(s):
        - UserClaimedTokens(_id,msg.sender,transferAmt) (src/Bonsai3Launchpad.sol#395)
Reentrancy in Bonsai3Launchpad.userClaimTokens(uint256)
(src/Bonsai3Launchpad.sol#376-407):
        External calls:
        - _endSale(saleLocalObj) (src/Bonsai3Launchpad.sol#383)
                - saleLocalObj.escrowContract.enableRefunds()
(src/Bonsai3Launchpad.sol#416)
                - saleLocalObj.escrowContract.close() (src/Bonsai3Launchpad.sol#419)
                - saleLocalObj.escrowContract.beneficiaryWithdraw()
(src/Bonsai3Launchpad.sol#420)
        - saleLocalObj.escrowContract.withdraw(address(msg.sender))
(src/Bonsai3Launchpad.sol#397)
        Event emitted after the call(s):
        - UserClaimedRefund(_id,msg.sender,saleLocalObj.totalUserPurchases[msg.sender])
(src/Bonsai3Launchpad.sol#398-402)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Bonsai3Launchpad.participateInSale(uint256,uint256) (src/Bonsai3Launchpad.sol#216-266)
uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= saleDetails[_id].endTime,Sale has ended)
(src/Bonsai3Launchpad.sol#227)
Bonsai3Launchpad._handleSaleState(Bonsai3Sale.NewSale) (src/Bonsai3Launchpad.sol#349-358)
uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(saleLocalObj.startTime <= block.timestamp,Sale has not
started yet) (src/Bonsai3Launchpad.sol#352-355)
Bonsai3Launchpad._endSale(Bonsai3Sale.NewSale) (src/Bonsai3Launchpad.sol#409-426) uses
timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= saleLocalObj.endTime,Sale time has not
elapsed) (src/Bonsai3Launchpad.sol#410-413)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Bonsai3Launchpad.setUp(IFactory,IFeeManager,address) (src/Bonsai3Launchpad.sol#81-91)
compares to a boolean constant:
        -require(bool,string)(_setup == false,contract has already been setup)
(src/Bonsai3Launchpad.sol#86)
Bonsai3Launchpad.contractTurnedOn() (src/Bonsai3Launchpad.sol#93-96) compares to a boolean
constant:
        -require(bool,string)(_setup == true,Please Start Contract)
(src/Bonsai3Launchpad.sol#94)
FeeManager.whenPaused() (src/FeeManager.sol#132-135) compares to a boolean constant:
        -require(bool,string)(__paused == true,not paused.) (src/FeeManager.sol#133)
FeeManager.notPaused() (src/FeeManager.sol#137-140) compares to a boolean constant:
        -require(bool,string)(__paused != true,not paused.) (src/FeeManager.sol#138)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
CombinedEscrow.burnAfterReading() (src/Libraries/CombinedEscrow.sol#152-154) is never used
and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
```

```
Pragma version>=0.8.19 (src/Bonsai3Launchpad.sol#2) necessitates a version too recent to
be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.19 (src/Disperse.sol#7) necessitates a version too recent to be
trusted. Consider deploying with 0.8.18.
Pragma version>=0.8.19 (src/FeeManager.sol#2) necessitates a version too recent to be
trusted. Consider deploying with 0.8.18.
Pragma version>=0.8.19 (src/Interfaces/IBonsai3Launchpad.sol#2) necessitates a version too
recent to be trusted. Consider deploying with 0.8.18.
Pragma version>=0.8.19 (src/Interfaces/ICombinedEscrow.sol#2) necessitates a version too
recent to be trusted. Consider deploying with 0.8.18.
Pragma version>=0.8.19 (src/Interfaces/IFeeManager.sol#2) necessitates a version too
recent to be trusted. Consider deploying with 0.8.18.
Pragma version>=0.8.19 (src/Libraries/Bonsai3Sale.sol#2) necessitates a version too recent
to be trusted. Consider deploying with 0.8.18.
Pragma version>=0.8.19 (src/Libraries/CombinedEscrow.sol#2) necessitates a version too
recent to be trusted. Consider deploying with 0.8.18.
Pragma version>=0.8.19 (src/Libraries/Sale.sol#2) necessitates a version too recent to be
trusted. Consider deploying with 0.8.18.
Pragma version>=0.8.19 (src/Libraries/SelfDestruct.sol#2) necessitates a version too
recent to be trusted. Consider deploying with 0.8.18.
Pragma version>=0.8.19 (src/Libraries/Token.sol#2) necessitates a version too recent to be
trusted. Consider deploying with 0.8.18.
Pragma version>=0.8.19 (src/Libraries/TokenDetails.sol#2) necessitates a version too
recent to be trusted. Consider deploying with 0.8.18.
Pragma version>=0.8.19 (src/TokenFactory.sol#2) necessitates a version too recent to be
trusted. Consider deploying with 0.8.18.
solc-0.8.19 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidi
ty
INFO:Detectors:
Low level call in
FeeManager._handleTokenPayment(uint256,IFeeManager.FeeTypes,uint256,address)
(src/FeeManager.sol#164-193):
        - (sent) = address(__vault).call{value: _purchaseAmount}()
(src/FeeManager.sol#179)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Bonsai3Launchpad (src/Bonsai3Launchpad.sol#30-541) should inherit from IBonsai3Mod
(src/Interfaces/IBonsai3Launchpad.sol#4-83)
Disperse (src/Disperse.sol#14-37) should inherit from Destroy
(src/Libraries/SelfDestruct.sol#4-13)
Factory (src/TokenFactory.sol#9-41) should inherit from IFactory
(src/Bonsai3Launchpad.sol#18-28)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance
INFO:Detectors:
Parameter Bonsai3Launchpad.setUp(IFactory,IFeeManager,address)._factory
(src/Bonsai3Launchpad.sol#82) is not in mixedCase
Parameter Bonsai3Launchpad.setUp(IFactory,IFeeManager,address)._feemanager
(src/Bonsai3Launchpad.sol#83) is not in mixedCase
Parameter Bonsai3Launchpad.setUp(IFactory,IFeeManager,address)._vault
(src/Bonsai3Launchpad.sol#84) is not in mixedCase
Parameter Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,address)._id
(src/Bonsai3Launchpad.sol#117) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,address)._totalSupply
(src/Bonsai3Launchpad.sol#118) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,address)._tokenName
(src/Bonsai3Launchpad.sol#119) is not in mixedCase
```

```
Parameter
Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,address)._tokenTicker
(src/Bonsai3Launchpad.sol#120) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewToken(uint256,uint256,string,string,uint8,address)._template
(src/Bonsai3Launchpad.sol#121) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address)._id (src/Bonsai3Launchpad.sol#175) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address)._totalSaleSupply (src/Bonsai3Launchpad.sol#176) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address)._presalePrice (src/Bonsai3Launchpad.sol#177) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address)._softCap (src/Bonsai3Launchpad.sol#178) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address)._startTime (src/Bonsai3Launchpad.sol#179) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address)._endTime (src/Bonsai3Launchpad.sol#180) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address)._saleToken (src/Bonsai3Launchpad.sol#181) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address)._purchaseToken (src/Bonsai3Launchpad.sol#182) is not in mixedCase
Parameter
Bonsai3Launchpad.createNewSale(uint256,uint256,uint256,uint256,uint256,uint256,address,add
ress,address)._userWallet (src/Bonsai3Launchpad.sol#183) is not in mixedCase
Parameter Bonsai3Launchpad.participateInSale(uint256,uint256)._id
(src/Bonsai3Launchpad.sol#217) is not in mixedCase
Parameter Bonsai3Launchpad.participateInSale(uint256,uint256)._purchaseAmount
(src/Bonsai3Launchpad.sol#218) is not in mixedCase
Parameter Bonsai3Launchpad.userClaimTokens(uint256)._id (src/Bonsai3Launchpad.sol#376) is
not in mixedCase
Parameter Bonsai3Launchpad.getListOfUsers(uint256)._id (src/Bonsai3Launchpad.sol#429) is
not in mixedCase
Parameter Bonsai3Launchpad.getSaleProgress(uint256)._id (src/Bonsai3Launchpad.sol#434) is
not in mixedCase
Parameter Bonsai3Launchpad.getParticipatedAmount(uint256)._id
(src/Bonsai3Launchpad.sol#447) is not in mixedCase
Parameter Bonsai3Launchpad.getUserPurchaseRecord(uint256,address)._id
(src/Bonsai3Launchpad.sol#454) is not in mixedCase
Parameter Bonsai3Launchpad.getTokenAllotmentForSaleByUser(uint256,address)._id
(src/Bonsai3Launchpad.sol#461) is not in mixedCase
Parameter Bonsai3Launchpad.getSalePurchaseToken(uint256)._id
(src/Bonsai3Launchpad.sol#467) is not in mixedCase
Parameter Bonsai3Launchpad.getTokenSaleAddress(uint256)._id (src/Bonsai3Launchpad.sol#471)
is not in mixedCase
Parameter Bonsai3Launchpad.getPreSalePrice(uint256)._id (src/Bonsai3Launchpad.sol#475) is
not in mixedCase
Parameter Bonsai3Launchpad.getSaleNumbers(uint256)._id (src/Bonsai3Launchpad.sol#480) is
not in mixedCase
Parameter Bonsai3Launchpad.getTokenDetails(uint256)._id (src/Bonsai3Launchpad.sol#504) is
not in mixedCase
Parameter Bonsai3Launchpad.getTokenAddresses(uint256)._id (src/Bonsai3Launchpad.sol#514)
is not in mixedCase
```

```
Parameter Bonsai3Launchpad.updateFactory(address)._newFactory
(src/Bonsai3Launchpad.sol#531) is not in mixedCase
Parameter Bonsai3Launchpad.updateFeeManager(address)._feeManager
(src/Bonsai3Launchpad.sol#537) is not in mixedCase
Variable Bonsai3Launchpad.__deployer (src/Bonsai3Launchpad.sol#69) is not in mixedCase
Variable Bonsai3Launchpad.__vault (src/Bonsai3Launchpad.sol#70) is not in mixedCase
Variable Bonsai3Launchpad._tokenFactory (src/Bonsai3Launchpad.sol#73) is not in mixedCase
Variable Bonsai3Launchpad._setup (src/Bonsai3Launchpad.sol#74) is not in mixedCase
Variable FeeManager.__paused (src/FeeManager.sol#11) is not in mixedCase
Variable FeeManager.__vault (src/FeeManager.sol#14) is not in mixedCase
Parameter Factory.deployToken(string,string,uint256,uint256,address)._name
(src/TokenFactory.sol#15) is not in mixedCase
Parameter Factory.deployToken(string,string,uint256,uint256,address)._ticker
(src/TokenFactory.sol#16) is not in mixedCase
Parameter Factory.deployToken(string,string,uint256,uint256,address)._supply
(src/TokenFactory.sol#17) is not in mixedCase
Parameter Factory.deployToken(string,string,uint256,uint256,address)._template
(src/TokenFactory.sol#18) is not in mixedCase
Parameter Factory.deployToken(string,string,uint256,uint256,address)._from
(src/TokenFactory.sol#19) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-nami
ng-conventions
INFO:Detectors:
FeeManager.constructor(address,address) (src/FeeManager.sol#56-114) uses literals with too
many digits:
        - fees[IFeeManager.FeeTypes.TokenTemplate1] =
FeeObject(0x01,IFeeManager.FeeTypes.TokenTemplate1,880000000000000,_vault,(address(0)))
(src/FeeManager.sol#71-77)
FeeManager.constructor(address,address) (src/FeeManager.sol#56-114) uses literals with too
many digits:
        - fees[IFeeManager.FeeTypes.TokenTemplate2] =
FeeObject(0x02,IFeeManager.FeeTypes.TokenTemplate2,50000000000000000,_vault,((address(0)))
) (src/FeeManager.sol#80-86)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
FeeManager.owner (src/FeeManager.sol#13) should be constant
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-b
e-declared-constant
INFO:Detectors:
FeeManager.__vault (src/FeeManager.sol#14) should be immutable
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-b
e-declared-immutable
```

Unit Test Coverage

```
Running 1 test for test/TokenFactory.t.sol:TestTokenFactory
[PASS] testMint() (gas: 3959889)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.50s
```

```
Running 11 tests for test/Bonsai3Launchpad.t.sol:Bonsai3Test
[PASS] testCreateManySales() (gas: 12272351)
[PASS] testCreateManyTokens() (gas: 4004024)
[PASS] testCreateNewSale() (gas: 2664168)
[PASS] testCreateNewSaleFullRun() (gas: 3018727)
[PASS] testCreateNewSaleParticipate() (gas: 3012291)
[PASS] testCreateToken() (gas: 983841)
[PASS] testFullRunNativePC() (gas: 2951161)
[PASS] testFullRunNativeRefund() (gas: 2950666)
[PASS] testFullRunTemplate2() (gas: 5079777)
[PASS] testManyParticipate() (gas: 13575823)
[PASS] testRealCase() (gas: 2952092)
Test result: ok. 11 passed; 0 failed; 0 skipped; finished in 10.01s
```

File	% Lines	% Statements	% Branches	% Funcs
script/Bonsai3Launchpad.s.sol	0.00% (0/10)	0.00% (0/12)	100.00% (0/0)	0.00% (0/1)
script/Disperse.s.sol	0.00% (0/3)	0.00% (0/3)	100.00% (0/0)	0.00% (0/1)
script/NewFactory.s.sol	0.00% (0/5)	0.00% (0/5)	100.00% (0/0)	0.00% (0/1)
script/NewFeeManager.s.sol	0.00% (0/4)	0.00% (0/4)	100.00% (0/0)	0.00% (0/1)
script/SingleToken.s.sol	0.00% (0/4)	0.00% (0/4)	100.00% (0/0)	0.00% (0/1)
script/UpgradeBonsai.s.sol	0.00% (0/6)	0.00% (0/7)	100.00% (0/0)	0.00% (0/1)
src/Bonsai3Launchpad.sol	77.21% (105/136)	77.24% (112/145)	35.71% (20/56)	51.43% (18/35)
src/Disperse.sol	0.00% (0/13)	0.00% (0/21)	0.00% (0/8)	0.00% (0/3)
src/FeeManager.sol	59.52% (25/42)	62.50% (30/48)	15.00% (3/20)	38.46% (5/13)
src/Libraries/CombinedEscrow.sol	74.19% (23/31)	71.88% (23/32)	6.67% (2/30)	50.00% (6/12)
src/TokenFactory.sol	86.67% (13/15)	76.19% (16/21)	50.00% (2/4)	50.00% (1/2)

For the **TokenFactory** contract, the uncovered lines are related to not calling **getAddress()** and not calling **deployToken()** with a template greater than 2.

```
34                  :            :          }
      35            :          0 :          return address(0);
      36            :            :      }
      37            :            :
      38            :            :      function getAddress() external
view returns (address) {
      39            :          0 :          return address(this);
      40            :            :      }
```

As for **FeeManager**, the missing test cases are related to:

1. **pausing** and **unpausing** the contract

```
     156                  :            :     function haltOperations() public
onlyAdmin notPaused {
     157                  :          0 :         __paused = true;
     158                  :            :     }
     159                  :            :
     160                  :            :     function serThePatientMadeIt()
public onlyAdmin whenPaused {
     161                  :          0 :         __paused = false;
     162                  :            :     }
```

2. fee payments made with ERC20 currencies were not tested, which is an important missing test.

```
180        [ #  # ]:         49 :             require(sent, "Failed to send
Ether");
     181                  :            :         } else {
     182                  :          0 :             IERC20 saleToken =
IERC20(fees[key].currency);
     183        [ #  # ]:          0 :             require(
     184                  :            :
saleToken.balanceOf(user) ≥ _purchaseAmount,
     185                  :            :                 "User does not have
the balance"
     186                  :            :             );
     187        [ #  # ]:          0 :             require(
     188                  :            :
saleToken.transferFrom(user, __vault, _purchaseAmount),
     189                  :            :                 "Token transfer
failed"
     190                  :            :             );
     191                  :            :         }
```

```
220                  :            :         } else {
     221                  :          0 :             _handleTokenPayment(
     222                  :            :
IERC20(invoices.currency).balanceOf(user),
     223                  :            :                 catalouge[key],
     224                  :            :
fees[catalouge[key]].amount,
     225                  :            :                 user
     226                  :            :             );
     227                  :          0 :             userPurchase.paid = true;
     228                  :          0 :             return true;
```

```
241                  :            :         } else {
     242                  :          0 :             userBalance =
IERC20(currency).balanceOf(payee) - balanceBefore;
```

```
     243                 :             :         }
```

3. unused functions which are: **`iFfeeExists`**, **`getFeeMapping`**, **`removeFee`** and **`getOwner`**.

```
255                  :             :     function iFfeeExists(uint8 feeId)
external view returns (bool) {
     256         [ #  # ]:             :         if (
     257                  :           0 :
fees[IFeeManager.FeeTypes(feeId)].feeType ==
     258                  :           0 :
IFeeManager.FeeTypes(feeId)
     259                  :             :         ) {
     260                  :           0 :             return true;
     261                  :             :         } else {
     262                  :           0 :             return false;
     263                  :             :         }
     264                  :             :     }
     265                  :             :
     266                  :             :     function getFeeMapping(
     267                  :             :         uint8 index
     268                  :             :     ) external view returns
(IFeeManager.FeeTypes) {
     269                  :           0 :         return catalouge[index];
     270                  :             :     }
     271                  :             :
     272                  :             :     // Remove a fee for a specific
address.
     273                  :             :     function
removeFee(IFeeManager.FeeTypes feeId) public onlyAdmin {
     274                  :           0 :         delete fees[feeId];
     275                  :             :     }
     281                  :             :     function getOwner() external view
returns (address) {
     282                  :           0 :         return owner;
     283                  :             :     }
     284                  :             :
     285                  :             :     function getVault() external view
returns (address) {
     286                  :           0 :         return __vault;
     287                  :             :     }
```

The contract's tests are lacking. It is therefore highly recommended to create suitable test-cases to cover what is missing.

Issue ID

BI3-64

Risk level

Severity: Medium, Likelihood: Medium

Description

Missing unit tests to cover:

- fees made in ERC20 currencies.
- **haltOperations**, **serThePatientMadeIt**, **iFfeeExists**, **getFeeMapping**, **removeFee** and **getOwner** functions.
- functions are reverting when they are supposed to.

Code location

```
bonsai3-smart-contracts/src/FeeManager.sol
```

Recommendation

Write additional test-cases to cover the uncovered code.

As for the **Bonsai3Launchpad** contract, the missing test cases are related to:

- **pausing** and **unpausing** the contract:

```
-   57                  :            :     function pause() public
    onlyRole(PAUSER_ROLE) {
-   58                  :          0 :         _pause();
-   59                  :            :     }
-   60                  :            :
-   61                  :            :     function unpause()
    public onlyRole(PAUSER_ROLE) {
-   62                  :          0 :         _unpause();
-   63                  :            :     }
```

- Not handling user purchases from sales with a **presalePrice** equaling zero:

```
-   279         [ + # ]:         10 :         if
    (saleLocalObj.presalePrice != 0) {
```

```
-       280                 :         10 :
    saleLocalObj.participatedAmount = _calculateTotalCost(
-       281                 :            :
    saleLocalObj.id,
-       282                 :            :                 _purchaseAmount
-       283                 :            :             );
-       284                 :            :         } else {
-       285                 :          0 :
    saleLocalObj.participatedAmount = _calculateUserTokens( // dutch
    auction price = 0
-       286                 :            :
    saleLocalObj.id,
-       287                 :            :                 _purchaseAmount
-       288                 :            :             );
-       289                 :            :         }
```

- Not calling the following functions: **`getListOfUsers`, `getSaleProgress`, `getListOfSalesByAddress`, `getUserPurchaseRecord`, `getTokenAllotmentForSaleByUser`, `getSalePurchaseToken`, `getSaleNumbers`, `getTokenDetails`, `getTokenAddresses`, `updateFactory`** and **`updateFeeManager`**.

As the missing test cases are important to guarantee the proper functioning of the contract, it is crucial to assess all the scenarios. Also, there were no unit tests covering reverts.

Issue ID

BI3-65

Risk level

Severity: Medium, Likelihood: Medium

Description

Missing unit tests to cover:

- Sales and purchases with a **`presalePrice`** equaling zero.
- **`pause`, `unpause`, `getListOfUsers`, `getSaleProgress`, `getListOfSalesByAddress`, `getUserPurchaseRecord`, `getTokenAllotmentForSaleByUser`, `getSalePurchaseToken`, `getSaleNumbers`, `getTokenDetails`, `getTokenAddresses`, `updateFactory`** and **`updateFeeManager`** functions.
- functions are reverting when they are supposed to.

Code location

```
bonsai3-smart-contracts/src/Bonsai3Launchpad.sol
```

Recommendation

Write additional test-cases to cover the uncovered code.

Disclaimer

While best efforts and precautions have been taken in the preparation of this document, The Arcadia Group and the Authors assume no responsibility for errors, omissions, or damages resulting from the use of the provided information. Additionally, Arcadia would like to emphasize that the use of Arcadia's services does not guarantee the security of a smart contract or set of smart contracts and does not guarantee against attacks. One audit on its own is not enough for a project to be considered secure; that categorization can only be earned through extensive peer review and battle testing over an extended period.