OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 33	2021-04-22	MrTeaThyme, Plemonade, Donut, Zapmore, Hebilicious	Audit First Findings
2.0	Total: 40	2021-04-29	MrTeaThyme, Plemonade, Donut, Zapmore, Hebilicious	Final Audit

Audit Notes

Audit Date	2021-04-15 - 2021-04-27
Auditor/Auditors	MrTeaThyme, Plemonade, Donut, Hebilicious
Auditor/Auditors Contact Information	tibereum-obelisk@protonmail.com
Notes	Specified code and contracts are audited for security flaws. UI/UX (website), logic, team, and tokenomics are not audited.
Audit Report Number	OB58159876

Disclaimer

This audit is not financial, investment, or any other kind of advice and is for informational purposes only. This report is not a substitute for doing your own research and due diligence. Obelisk is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Obelisk has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Obelisk is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. The audit is paid by the project but neither the auditors nor Obelisk has any other connection to the project and has no obligations other than to publish an objective report. Obelisk will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Obelisk assumes that the provided information and material was not altered, suppressed, or misleading. This report is published by Obelisk, and Obelisk has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Obelisk.

Obelisk Auditing

Binance Smart Chain (BSC) launched in fall 2020 as a faster and cheaper alternative to Ethereum. Since then BSC has seen exponential growth to a point where there is a multitude of new projects created every day. In a fast phased world like this, there will also be an enormous amount of scams. The scams have become so elaborate that it's hard for the common investor to trust a project, even though it could be legit. We saw a need for creating high-quality audits at a fast phase to keep up with the constantly expanding market. With the Obelisk stamp of approval, a legitimate project can easily grow its user base exponentially in a world where trust means everything. Obelisk Auditing consists of a group of security experts that specialize in security and structural operations, with previous work experience from among other things, PricewaterhouseCoopers. All our audits will always be conducted by at least two independent auditors for maximum security and professionalism.

As a comprehensive security firm, Obelisk provides all kinds of audits and project assistance.

Table of Content

Project Information

Project Name	Strainz
Description	The first weed NFT ecosystem and all-inclusive marketplace on Binance Smart Chain.
Contact	@mrniach
Contact information	@mrniach on TG
Token Name(s)	Strainz, Seedz
Token Short	STRAINZ, SEEDZ
Contract(s)	See Appendix A
Code Language	Solidity
Chain	Binance Smart Chain (BSC)

Executive Summary

The audit of Strainz was conducted by four of Obelisks' security experts between the 15th of April 2021 and the 27th of April 2021. The contracts audited are not published yet.

After finishing the full audit, Obelisk auditing can say that there were some security issues with the audited contracts from Strainz. The project team did updates on the project contracts and solved some of the issues, and commented on other issues found. Obelisk has not reviewed the UI/UX, logic, team, or tokenomics of the project.

Please read the full document for a complete understanding of the audit.

Summary Table

Audited Part	Severity	Note
Unbounded Loop will block operation at Scale	High	Mitigated
Unbounded Loop will block operation at Scale	High	Mitigated
Unbounded Loop	Low	See Comment
Unsafe Random	Medium	Mitigated
Loop Over unbounded array/Optimization	Low	See Comment
Checks-Effects-Interactions Violation	Low	Mitigated
Use safeTransferFrom	Low	Mitigated
No timelock to modify contract values	High	See Comment
Modifiable private values	Low	Mitigated
Fertilizing and composting cost and reward	Informational	Mitigated
Required grow rate for breeding	Informational	N/A
Watering and harvesting can only be done together	Informational	N/A
The harvestable amount is calculated incorrectly	Informational	N/A
Migration from V1 can be blacklisted	Medium	See Comment
Accessories are not migrated	Medium	See Comment
Harvestable fertilizer amount is calculated incorrectly	Low	N/A
Unbounded loop	Informational	See Comment

Migrate Permissions	Medium	N/A
Owner MintNFT	Medium	N/A
Could Not Verify Individual Contracts	Informational	Mitigated
No Timelock Present	High	See Comment

Introduction

Obelisk was commissioned by Strainz on the 14th of April 2021 to conduct a comprehensive audit of the Strainz project. The following audit was conducted between the 15th of April 2021 and the 27th of April 2021 and delivered on the 29rd of April 2021. Four of Obelisks security experts went through the related contracts using industry standards to find if any vulnerabilities could be exploited.

The comprehensive test was conducted in a specific test environment that utilized exact copies of the published contract. The auditors also conducted a manual visual inspection of the code to find security flaws that automatic tests would not find.

While conducting the audit, the Obelisk security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Obelisk assesses the risks and assigns a risk level to each section together with an explanatory comment.

The audited contracts were not yet published during the audit. There was an on-chain analysis conducted after the contracts were published to match the unpublished contracts with the published contracts. While conducting the audit of Strainz, our auditors found a mixture of issues ranging from low to high. The project team has commented on a number of these and solved some others. Please see each issue separately to get a deeper understanding of the issue and its resolution, if any. Take note that the comments from the project team are their opinion and not the opinion of Obelisk.

Findings

Manual Analysis

Unbounded Loop will block operation at Scale

SEVERITY	Mitigated (High)
LOCATION	-

```
function updateRewards() private {
        uint timeSinceLastUpdate = block.timestamp - timeOfLastUpdate;
        if (timeSinceLastUpdate == 0) {
            return;
        }

        for (uint i = 0; i < numberOfPools; i++) {
            Pool storage pool = pools[i];
            uint lpSupply = pool.lpToken.totalSupply();
            uint rewardsLeft = pool.amountToDistribute - pool.amountDistributed;
            uint rewardsSinceLastUpdate = (timeSinceLastUpdate * (rewardsLeft / 20)) / 1 weeks;

            for (uint j = 0; j < pool.stakeholders.length; j++) {
                address stakeHolder = pool.stakeholders[j];
                Stake storage stake = pool.stakes[stakeHolder];
                uint amount = rewardsSinceLastUpdate * stake.depositedAmount / lpSupply;

                stake.claimableAmount += amount;
                pool.amountDistributed += amount;
            }
        }
        timeOfLastUpdate = block.timestamp;

    }
```

DESCRIPTION	In the SeedzToken contract a deposit/withdrawal functions called the updateRewards function. The updateRewards function loops through every pool and every user to update the rewards. This is a nested iteration and such could become unusable with a high enough number of pools and users. This would lock those functions from functioning and essentially locking users funds as no emergency withdrawal.
RECOMMENDATION	Find an alternative rewards tracking system that does not use iteration.
MITIGATED/COMMENT	The project team changed the implementation to the SeedzToken contract to be more efficient and not rely on loops.

Unbounded Loop will block operation at Scale

SEVERITY	Mitigated (High)
LOCATION	-

```
function addStakeHolder(Pool storage pool, address stakeholder) private {
        bool alreadyInPool = false;
        for (uint i = 0; i < pool.stakeholders.length; i++) {
            if (pool.stakeholders[i] == stakeholder) {
                alreadyInPool = true;
            }
        }

        if (!alreadyInPool) {
            pool.stakeholders.push(stakeholder);
        }
    }
```

DESCRIPTION	The function iterates through all the users to check if a user exists. This makes the operation trend towards an infinite gas cost as more users are added. This function could become unusable if enough users are added.
RECOMMENDATION	Implement a mapping with users in a pool instead as there won't be a need to check if stakeholders are inside the pool already
MITIGATED/COMMENT	The project team changed the implementation to the SeedzToken contract to be more efficient and not rely on loops.

Unbounded Loop

SEVERITY	Mitigated(Low)
LOCATION	-

```
    function getHarvestableAccessoryAmount(uint strainId, uint timeSinceLastHarvest) public view
returns (uint) {
        uint[] memory accessoryIds = getAccessoriesByStrainId(strainId);

        uint accessoryBonus = 0;
        for (uint i = 0; i < accessoryIds.length; i++) {
            uint accessoryType = accessoryTypeByTokenId[accessoryIds[i]];
            uint boost = growBonusForType[accessoryType];
            uint timeOfAttachment = timeOfLastAttachment[accessoryIds[i]];
            if (timeOfAttachment == 0) {
                continue;
            }
            uint attachTime = min(block.timestamp - timeOfAttachment, timeSinceLastHarvest);

            accessoryBonus += attachTime * boost / 1 days;
        }
        return accessoryBonus;
    }
```

```
    // detach all (compost)
    function detachAll(uint strainId) public onlyStrainzNFT {
        uint[] memory accessoryIds = accessoriesByStrainId[strainId];
        for (uint i = 0; i < accessoryIds.length; i++) {
            detachAccessory(accessoryIds[i], strainId);
        }
    }
```

```
  function sameTypeAlreadyAttached(uint[] storage array, uint newType) private view returns (bool) {
        for (uint i = 0; i < array.length; i++) {
            uint existingType = accessoryTypeByTokenId[array[i]];
            if (existingType == newType) {
                return true;
            }
        }
        return false;
    }
```

```
// generates accessories based on parents/fertilizer
function breedAccessories(uint strain1Id, uint strain2Id, uint newStrainId) public onlyStrainzNFT {
    uint[] storage strain1Accessories = accessoriesByStrainId[strain1Id];
    uint[] storage strain2Accessories = accessoriesByStrainId[strain2Id];

    for (uint i = 1; i <= numberOfAccessoryTypes; i++) {
        bool hasParent1 = sameTypeAlreadyAttached(strain1Accessories, i);
        bool hasParent2 = sameTypeAlreadyAttached(strain2Accessories, i);
        if (hasParent1 && hasParent2){
            mintAccessory(i, newStrainId);
        }

    }

}
```

```
function indexOf(uint[] storage array, uint tokenId) private view returns (int) {
    for (uint i = 0; i < array.length; i++) {
        if (array[i] == tokenId) {
            return int(i);
        }
    }
    return - 1;
}
```

DESCRIPTION	Functions iterate through an unbounded array. These functions could reach the max gas cap if the array gets large enough.
RECOMMENDATION	Since it doesn't cost gas if used only externally, write those functions as external if they're intended to be used externally. Implementing a cap to ensure the max gas limit isn't reached is another option. Avoid loops as much as possible while working with solidity. if it's necessary to loop over an unbounded array then plan for it to take multiple blocks by saving the state it's in.
MITIGATED/COMMENT	Project team comment: "this won't be a problem, as the array is bounded to the amount of different accessories which is 3 and will stay <5 for sure if we even will add new ones" Obelisk: "There is nothing that stops new accessories from being added in the code. It's suggested to set a limit or to

	use a different implementation.”

Unsafe Random

SEVERITY	Mitigated (Medium)
LOCATION	-

```
function random(uint nonce) private view returns (uint) {
    return uint(keccak256(abi.encodePacked(block.difficulty, block.timestamp, nonce))) % 25500;
}
```

DESCRIPTION	Random function is used with incentives. The random function is not secure enough and thus the following scenarios could potentially happen. **1.** An attacker can use a contract to pre-calculate these values and send a transaction with a known favorable result. **2.** Do not rely on predictable inputs such as block.timestamp as a source of randomness as it can be manipulated by miners to a certain extent **3.** Validators/miners could choose to skip blocks if there are enough incentives (to try to get a good block.difficulty)
RECOMMENDATION	We recommend using an oracle for randomness, or to use an off-chain solution, as on-chain randomness is extremely hard to implement in a deterministic system like the EVM.
MITIGATED/COMMENT	Project team comment: ”removed all relevant randoms (the random left is for the look of the plant, and gives no monetary benefits)”

Loop Over unbounded array/Optimization

SEVERITY	Low
LOCATION	*StrainzNFT.sol => Line 217*

```
    function migrate(address user) public onlyMaster returns (uint) {


        uint numberOfStrainz = strainzV1NFT.balanceOf(user);

        uint sumToHarvest = 0;
        //migrate NFT
        if (numberOfStrainz > 0) {
            uint[] memory tokenIds = new uint[](numberOfStrainz);
            for (uint i = 0; i < numberOfStrainz; i++) {
                tokenIds[i] = strainzV1NFT.tokenOfOwnerByIndex(user, i);
            }
            for (uint i = 0; i < tokenIds.length; i++) {
                StrainMetadata memory strain = getV1Strain(tokenIds[i]);
                strainzV1NFT.transferFrom(user, address(this), tokenIds[i]); // burn v1


                if (!blacklist[tokenIds[i]]) {
                        uint timeSinceLastHarvest = block.timestamp - strain.lastHarvest;
                        uint amountToHarvest = (strain.growRate * 255 * timeSinceLastHarvest) / 24 weeks;
                        // old formular
                        sumToHarvest += amountToHarvest;
                        mintTo(user, strain.prefix, strain.postfix, strain.dna, strain.generation, max(16,
strain.growRate));

                    }
                }
            }


        uint amountOfStrainzV1Tokens = strainzV1Token.balanceOf(user);
        strainzV1Token.transferFrom(user, address(this), amountOfStrainzV1Tokens);
        sumToHarvest += amountOfStrainzV1Tokens;

        return blacklistedUser[user] ? 0 : sumToHarvest;
    }
```

DESCRIPTION	1.Make sure this function will always be within the max gas limit. 2. The second loop is unnecessary here as it could utilize the first loop.
RECOMMENDATION	Consider implementing a way for users to choose what token ids they want to migrate so it's possible to split the migration over multiple transactions. The code above could be refactored to have only one iteration.
MITIGATED/COMMENT	The project team:"removed unnecessary loop". The project team has locked numberOfStrainz to 50 to mitigate the out of gas problem. So it's very unlikely to cause a problem

Checks-Effects-Interactions Violation

SEVERITY	Mitigated(Low)
LOCATION	*SeedzToken.sol => Line 195*

```
function emergencyWithdraw(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    pool.lpToken.transfer(address(msg.sender), user.amount);
    emit EmergencyWithdraw(msg.sender, _pid, user.amount);
    user.amount = 0;
    user.rewardDebt = 0;
}
```

DESCRIPTION	The function does not follow the Checks-Effects-Interactions pattern. To reduce the attack surface of reentrancy please follow the Checks-Effects-Interactions pattern.
RECOMMENDATION	Consider setting user amount and rewardebt to 0 before calling the transfer function.
MITIGATED/COMMENT	The project team implemented the recommended fix. This issue existed in the SeedzToken contract.

Use safeTransferFrom

SEVERITY	Mitigated(Low)
LOCATION	-

```
    function openERC721Trade(address nftContractAddress, uint tokenId, uint price) public {
        IERC721 nftContract = IERC721(nftContractAddress);
        require(nftContract.ownerOf(tokenId) == msg.sender);
        _tradeCounter.increment();
        nftContract.transferFrom(msg.sender, address(this), tokenId);
        uint id = _tradeCounter.current();
        erc721Trades[id] = ERC721Trade(id, msg.sender, nftContractAddress, tokenId, price,
TradeStatus.Open, address(0));

        emit ERC721TradeStatusChange(id, TradeStatus.Open);
    }
```

```
    function executeERC721Trade(uint tradeId) public {
        ERC721Trade memory trade = erc721Trades[tradeId];
        require(trade.status == TradeStatus.Open);
        uint marketPlaceShare = trade.strainzTokenPrice / 50;

        master.strainzToken().marketPlaceBurn(msg.sender, marketPlaceShare);

        master.strainzToken().transferFrom(msg.sender, trade.poster, trade.strainzTokenPrice -
marketPlaceShare);
        IERC721 nftContract = IERC721(trade.nftContractAddress);
        nftContract.transferFrom(address(this), msg.sender, trade.tokenId);

        erc721Trades[tradeId].status = TradeStatus.Closed;
        erc721Trades[tradeId].buyer = msg.sender;
        emit ERC721TradeStatusChange(tradeId, TradeStatus.Closed);
    }
```

```
    function cancelERC721Trade(uint tradeId) public {
        ERC721Trade memory trade = erc721Trades[tradeId];
        require(msg.sender == trade.poster);
        require(trade.status == TradeStatus.Open);
        IERC721 nftContract = IERC721(trade.nftContractAddress);
        nftContract.transferFrom(address(this), trade.poster, trade.tokenId);

        erc721Trades[tradeId].status = TradeStatus.Cancelled;
        emit ERC721TradeStatusChange(tradeId, TradeStatus.Cancelled);
    }
```

DESCRIPTION	Using transferFrom is discouraged because if the NFT is transferred to a contract that can't handle erc721 tokens it will be lost.
RECOMMENDATION	Consider using safeTransferFrom to ensure that the receiver can handle erc721 tokens
MITIGATED/COMMENT	The project team has implemented safeTransferFrom on the needed function as it can be assumed that the address that sent the nft and the contract itself is able to handle erc721 tokens. This issue existed in the StrainzMarketplace contract.

No timelock to modify contract values

SEVERITY	High
LOCATION	*StrainzMaster.sol => line 71*

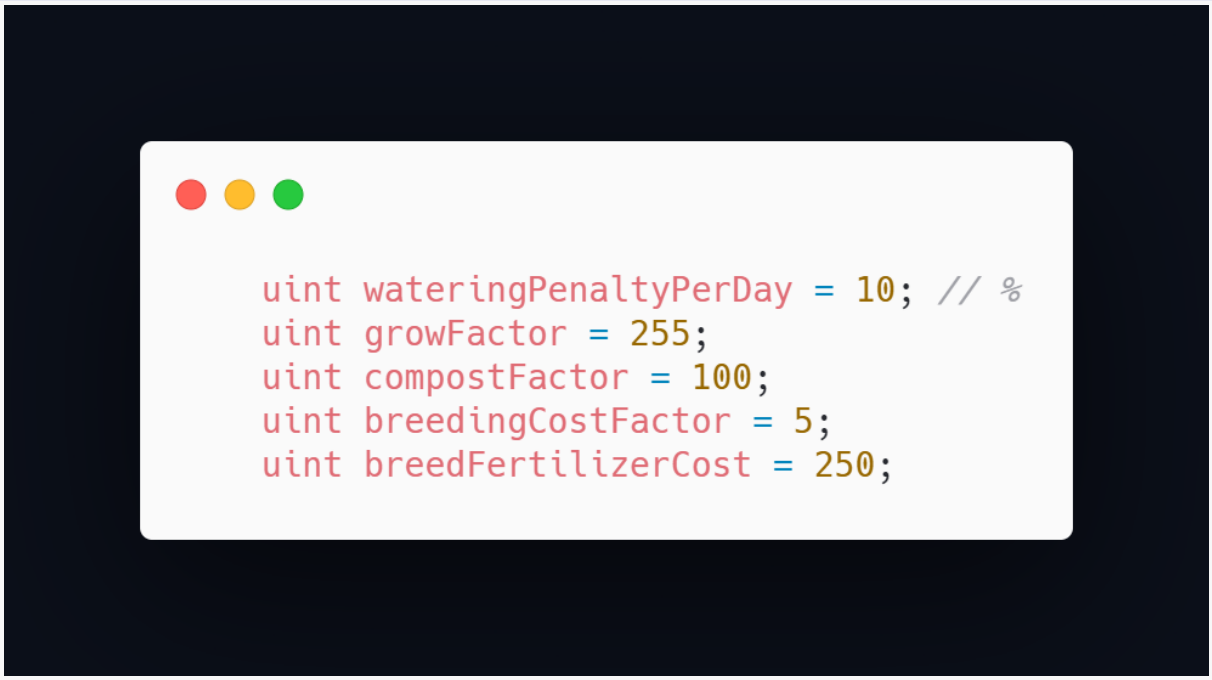
```
function addSeedzPool(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyOwner {
        seedzToken.add(_allocPoint, _lpToken, _withUpdate);
    }
    function setSeedzPool(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {
        seedzToken.set(_pid, _allocPoint, _withUpdate);
    }

    function setSeedzPerBlock(uint _seedzPerBlock) public onlyOwner {
        seedzToken.setSeedzPerBlock(_seedzPerBlock);
    }
```

DESCRIPTION	A malicious actor in control of the MasterStrainz contract can change the contract's values at any time. Of particular note is the ability to add and change the reward rate of SEEDZ pools. The malicious actor can modify the pools to give themselves an unlimited quantity of SEEDZ and use that to dilute the liquidity pool for other users.
RECOMMENDATION	Set the contract owner to a Timelock contract to give users time to react to any changes to the contract runtime values.
MITIGATED/COMMENT	The project team: "We will make a timelock contract later and transfer the ownership, right now the keys are on a hardware wallet"

Modifiable private values

SEVERITY	Mitigated(Low)
LOCATION	-



```
uint wateringPenaltyPerDay = 10; // %
uint growFactor = 255;
uint compostFactor = 100;
uint breedingCostFactor = 5;
uint breedFertilizerCost = 250;
```

DESCRIPTION	Modifiable values should be public or have custom getters for users to easily be able to check the current value of the variables.
RECOMMENDATION	Change values to public.
MITIGATED/COMMENT	The project team has implemented the recommended changes in the strainzNFT Contract

Fertilizing and composting cost and reward

SEVERITY	Mitigated (Informational)
LOCATION	*StrainzNFT.sol -> lines 26 and 28* *StrainzNFT.sol -> line 70* *StrainzNFT.sol -> line 103* *SeedzToken.sol -> line 24* *SeedzToken.sol -> line 22*

```
uint compostFactor = 100;
uint breedingCostFactor = 5;
uint breedFertilizerCost = 250;
```

```
function breed(uint _first, uint _second, bool breedFertilizer) public {
        require(ownerOf(_first) == msg.sender && ownerOf(_second) == msg.sender);
        StrainMetadata storage strain1 = strainData[_first];
        StrainMetadata storage strain2 = strainData[_second];

        uint strainzCost = (strain1.breedingCost + strain2.breedingCost) / 2;

        // Burn cost
        master.strainzToken().breedBurn(msg.sender, strainzCost);

        uint newStrainId = mixBreedMint(strain1, strain2, breedFertilizer);
        uint averageGrowRate = (strain1.growRate + strain2.growRate) / 2;
        // Burn fertilizer cost
        if (breedFertilizer && averageGrowRate >= 128) {
////////////////////////////////////////////////////////////////////////////
            master.seedzToken().breedBurn(msg.sender, breedFertilizerCost);
 ////////////////////////////////////////////////////////////////////////////
            master.strainzAccessory().breedAccessories(strain1.id, strain2.id, newStrainId);
        }


        emit Breed(strain1.id, strain2.id, newStrainId);

    }
```

```
  function compost(uint strainId) public {
        require(ownerOf(strainId) == msg.sender);
        StrainMetadata storage strain = strainData[strainId];
        master.strainzAccessory().detachAll(strainId);
        _burn(strainId);
////////////////////////////////////////////////////////////////////////////
        master.seedzToken().compostMint(msg.sender, strain.growRate * compostFactor / 100);
////////////////////////////////////////////////////////////////////////////
        emit Composted(strainId);
    }
```

```
    uint public growFertilizerCost = 500;
```

```
    function buyGrowFertilizer(uint plantId) public {
        uint cost = growFertilizerCost;

        require(balanceOf(msg.sender) >= cost);
        require(lastTimeGrowFertilizerUsedOnPlant[plantId] + 1 weeks < block.timestamp);
////////////////////////////////////////////////////////////////////////////
        _burn(msg.sender, cost);
////////////////////////////////////////////////////////////////////////////
        lastTimeGrowFertilizerUsedOnPlant[plantId] = block.timestamp;
        emit FertilizerBought(msg.sender, plantId);
    }
```

DESCRIPTION	18 decimal places are used in SEEDZ. However, functions that mint and burn SEEDZ do not take this into account.
RECOMMENDATION	Change the noted values to reflect the correct number of decimals
MITIGATED/COMMENT	The project team:”will provide a bonus for one week since time of attachment, (can be harvested multiple times in between). start and end calculate the time range in which an fertilizer was valid since the last harvest. end is always greater than or equal start because of this” In deployed code: Values were adjusted.

Required grow rate for breeding

SEVERITY	Informational
LOCATION	*strainzNFT.sol -> line 69*

```
function breed(uint _first, uint _second, bool breedFertilizer) public {
        require(ownerOf(_first) == msg.sender && ownerOf(_second) == msg.sender);
        StrainMetadata storage strain1 = strainData[_first];
        StrainMetadata storage strain2 = strainData[_second];

        uint strainzCost = (strain1.breedingCost + strain2.breedingCost) / 2;

        // Burn cost
        master.strainzToken().breedBurn(msg.sender, strainzCost);

        uint newStrainId = mixBreedMint(strain1, strain2, breedFertilizer);
        uint averageGrowRate = (strain1.growRate + strain2.growRate) / 2;
        // Burn fertilizer cost

////////////////////////////////////////////////////////////////////////////
        if (breedFertilizer && averageGrowRate >= 128) {
////////////////////////////////////////////////////////////////////////////
            master.seedzToken().breedBurn(msg.sender, breedFertilcompostMintizerCost);
            master.strainzAccessory().breedAccessories(strain1.id, strain2.id, newStrainId);
        }


        emit Breed(strain1.id, strain2.id, newStrainId);

    }
```

DESCRIPTION	Medium suggests the average growth rate of both parents should be at least 150 but actual in the contract is 128
RECOMMENDATION	Update the medium or code to match.
MITIGATED/COMMENT	N/A

Watering and harvesting can only be done together

SEVERITY	Informational
LOCATION	StrainzNFT.sol -> line 132-142

```
function harvestAndWaterAll() public {
      uint numberOfTokens = balanceOf(msg.sender);
      require(numberOfTokens > 0);
      uint sum = 0;
      for (uint i = 0; i < numberOfTokens; i++) {
          StrainMetadata storage strain = strainData[tokenOfOwnerByIndex(msg.sender, i)];
          sum += harvestableAmount(strain.id) - getWateringCost(strain.id);
          strain.lastHarvest = block.timestamp;
      }
      master.strainzToken().harvestMint(msg.sender, sum);
  }
```

DESCRIPTION	Harvesting and watering are only available through the noted function. The medium post suggests that harvesting and watering individual plants is possible.
RECOMMENDATION	Provide additional functionality or change the medium post. Note that currently, the contract assumes that watering and harvesting always happen simultaneously.
MITIGATED/COMMENT	N/A

The harvestable amount is calculated incorrectly

SEVERITY	Informational
LOCATION	StrainzNFT.sol -> 158-170

```
    function getAccumulatedHarvestAmount(StrainMetadata storage strain) private view returns (uint) {
        uint wateringRange = min(block.timestamp - strain.lastHarvest, 9 days);

        uint growRate = strain.growRate * 1647058824;

        uint harvestableSum = (((20 * growRate * wateringRange * 1 days) - (growRate * wateringRange *
wateringRange))) / (20 * 1 days * 1 days) / 1000000000;

        uint stagnationSum = 0;
        if (block.timestamp - strain.lastHarvest > 9 days) {
            stagnationSum = (block.timestamp - strain.lastHarvest + 9 days) * growRate * 10 /
100000000000 days;
        }
        return harvestableSum + stagnationSum;
    }
```

DESCRIPTION	The function used to calculate the harvest amount for an NFT is incorrect. While elsewhere, the contract assumes that growth rate changes in discrete steps, here the growth rate is assumed to change linearly. Further, on line 167, stagnationSum has a calculation error.
RECOMMENDATION	Change the growth rate to be consistently discrete or linear throughout the contract. Fix math errors. Provide commentary on the exact nature of the calculations.
MITIGATED/COMMENT	N/A

Migration from V1 can be blacklisted

SEVERITY	Medium
LOCATION	StrainzNFT.sol => line 205 StrainzMaster.sol => line 67

```
function blacklistCheaters(uint[] calldata tokens, address[] calldata users) public onlyOwner {
    strainzNFT.blacklistCheaters(tokens, users);
}
```

```
function blacklistCheaters(uint[] calldata tokens, address[] calldata users) public onlyMaster {
    for (uint i = 0; i < tokens.length; i++) {
        blacklist[tokens[i]] = true;
    }
    for (uint i = 0; i < users.length; i++) {
        blacklistedUser[users[i]] = true;
    }
}
```

DESCRIPTION	A malicious actor in control of the master contract can blacklist any user or token from successfully migrating. However, the Strainz V1 tokens and NFTs will still be transferred away from the account.
RECOMMENDATION	Set the contract owner to a Timelock contract.
MITIGATED/COMMENT	The project team: “We will make a timelock contract later and transfer the ownership, right now the keys are on a hardware wallet”

Accessories are not migrated

SEVERITY	Medium
LOCATION	-

```
function migrate(address user) public onlyMaster returns (uint) {


        uint numberOfStrainz = strainzV1NFT.balanceOf(user);

        uint sumToHarvest = 0;
        //migrate NFT
        if (numberOfStrainz > 0) {
            uint[] memory tokenIds = new uint[](numberOfStrainz);
            for (uint i = 0; i < numberOfStrainz; i++) {
                tokenIds[i] = strainzV1NFT.tokenOfOwnerByIndex(user, i);
            }
            for (uint i = 0; i < tokenIds.length; i++) {
                StrainMetadata memory strain = getV1Strain(tokenIds[i]);
                strainzV1NFT.transferFrom(user, address(this), tokenIds[i]); // burn v1


                if (!blacklist[tokenIds[i]]) {
                        uint timeSinceLastHarvest = block.timestamp - strain.lastHarvest;
                        uint amountToHarvest = (strain.growRate * 255 * timeSinceLastHarvest) / 24 weeks;
                        // old formular
                        sumToHarvest += amountToHarvest;
                        mintTo(user, strain.prefix, strain.postfix, strain.dna, strain.generation, max(16,
strain.growRate));

                    }
                }
            }


        uint amountOfStrainzV1Tokens = strainzV1Token.balanceOf(user);
        strainzV1Token.transferFrom(user, address(this), amountOfStrainzV1Tokens);
        sumToHarvest += amountOfStrainzV1Tokens;

        return blacklistedUser[user] ? 0 : sumToHarvest;
    }
```

DESCRIPTION	Accessories are not migrated. Medium suggests that migration from V1 to V2 will preserve accessories.
RECOMMENDATION	Add migration of accessories.
MITIGATED/COMMENT	Accessory migration was added. Care should be taken to initialize the number of accessories correctly. This issue existed in the StrainzNFT contract. Deployed code: When migrating, accessory types 1, 2, and 3 are existing accessories. The number of accessory types should be initialized to reflect this. This was not checked because the contract was not verified.

Harvestable fertilizer amount is calculated incorrectly

SEVERITY	Low
LOCATION	SeedzToken.sol -> line 225

```
function getHarvestableFertilizerAmount(uint strainId, uint lastHarvest) public view returns (uint) {
      uint fertilizerBonus = 0;
      uint fertilizerAttachTime = lastTimeGrowFertilizerUsedOnPlant[strainId];
      if (fertilizerAttachTime > 0) {
          uint start = max(fertilizerAttachTime, lastHarvest);

////////////////////////////////////////////////////////////////////////////
          uint end = min(start + 1 weeks, block.timestamp);
////////////////////////////////////////////////////////////////////////////

          fertilizerBonus = (end - start) * growFertilizerBoost / 1 days;
      }
      return fertilizerBonus;
  }
```

DESCRIPTION	Fertilizers will provide a bonus for a week after the last harvest once fertilized. If the start is greater than the end this function will revert and prevent the harvesting of rewards.
RECOMMENDATION	Fix calculation of end and check for the correct time range.
MITIGATED/COMMENT	N/A

Unbounded loop

SEVERITY	Informational
LOCATION	-

```
function signerInArray(address signer, address[] memory signerArray) private pure returns (bool) {
    for (uint i = 0; i < signerArray.length; i++) {
        if (signerArray[i] == signer) {
            return true;
        }
    }
    return false;
}
```

DESCRIPTION	Looping through an unbounded array could cause these functions to run out of gas if there are enough signers.
RECOMMENDATION	Consider using a mapping instead of looping through all the signers if possible. A mapping will always map to a value compared to an array that could be uninitialized. A mapping lookup is executed in constant time. Another solution is to limit the signers array.
MITIGATED/COMMENT	The project team has stated that multisigvault contract are subject to change as it will be deployed at a later stage with more features.

Migrate Permissions

SEVERITY	Medium
LOCATION	StrainzMaster.sol line => 25

```
function migrate() public {
        require(migrationActive);
        uint amountToHarvest = strainzNFT.migrate(msg.sender);
        strainzToken.migrateMint(msg.sender, amountToHarvest);
    }

    function setMigration(bool active) public onlyOwner {
        migrationActive = active;
    }
```

DESCRIPTION	The owner address is able to disable the migration for all users at will.
RECOMMENDATION	Consider implementing a timelock to give users time to react as migrations disabling shouldn't be done drastically to give users time to migrate their funds.
MITIGATED/COMMENT	N/A

Owner MintNFT

SEVERITY	Informational/Medium
LOCATION	StrainzMaster.sol line => 56

```
    // mint promotional unique strainz (will get custom images)
    function mintPromotion(address receiver, string memory prefix, string memory postfix, uint dna)
public onlyMaster {
        mintTo(receiver, prefix, postfix, dna, 0, 255);
    }
```

DESCRIPTION	Owner is able to mint nfts by using the mint promotion function.
RECOMMENDATION	Consider implementing a timelock to give users a heads up when promotional nfts are going to be minted.
MITIGATED/COMMENT	N/A

Static Analysis

No Findings

On-Chain Analysis

Could Not Verify Individual Contracts

SEVERITY	Mitigated(Informational)
DESCRIPTION	Contracts are deployed but only StrainzMaster is verified. There is no risk of the incorrect contract being deployed; however, the remaining contracts should be verified to allow users to inspect the contracts individually.
RECOMMENDATION	Verify remaining contracts
MITIGATED/COMMENT	The project team has verified the smart contracts.

No Timelock Present

SEVERITY	High
DESCRIPTION	Refer to Manual Analysis for risks associated with the absence of a timelock.
RECOMMENDATION	Set the contract owner to a Timelock contract to give users time to react to any changes to the contract runtime values.
MITIGATED/COMMENT	The project team: “We will make a timelock contract later and transfer the ownership, right now the keys are on a hardware wallet”

Appendix A - Reviewed Documents

Document	Address
SeedzToken.sol	0x789597a72aff54fadea196c13fa1011c2e3c2a1e
StrainMetadata.sol	Interface used by StrainzNFT
StrainzAccessory.sol	0x526322ac7ddf1b7da4967d33a4b03f53ab757a59
StrainzDNA.sol	Inherited by StrainzNFT
StrainzMarketplace.sol	0xfdf8f3c8d34dc3f6bbc51098173acbd59e344548
StrainzMaster.sol	0xDBEA85Ce8d2BFfAA292E6def9B22a5DAa40d716E
StrainzMultisigFactory.sol	Not Deployed
StrainzMultisigVault.sol	Not Deployed
StrainzNFT.sol	0x196546f21c30f1b4b5846ed6a1bd2f7c6c2f6db5
StrainzToken.sol	0x18fcd272fcbb41cf048c89695dfeb6a8d298beb1

Appendix B - Risk Ratings

Risk	Description
High Risk	A fatal vulnerability that can cause immediate loss of Tokens / Funds
Medium Risk	A vulnerability that can cause some loss of Tokens / Funds
Low Risk	A vulnerability that can be mitigated
Informational	No vulnerability

Appendix C - Icons

Icon	Explanation
	Solved by Project Team
	Under Investigation of Project Team
	Unsolved

Appendix D - Testing Standard

An ordinary audit is conducted using these steps.

1. Gather all information
2. Conduct a first visual inspection of documents and contracts
3. Go through all functions of the contract manually (2 independent auditors)
    a. Discuss findings
4. Use specialized tools to find security flaws
    a. Discuss findings
5. Follow up with project lead of findings
6. If there are flaws, and they are corrected, restart from step 2
7. Write and publish a report

During our audit, a thorough investigation has been conducted employing both automated analysis and manual inspection techniques. Our auditing method lays a particular focus on the following important concepts:

- Ensuring that the code and codebase use best practices, industry standards, and available libraries.
- Testing the contract from different angles ensuring that it works under a multitude of circumstances.
- Analyzing the contracts through databases of common security flaws.

OBELISK

Part of Tibereum Group