CERTIK

Code Security Assessment

Boba Brewery

Feb 24th, 2022

Table of Contents

Summary

This report has been prepared for Boba Brewery to discover issues and vulnerabilities in the source code of the Boba Brewery project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

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

Additionally, this audit is based on a premise that all external contracts were implemented safely. And the following `sol` files are not within the scope of the audit:

- `@openzeppelin/contracts/token/ERC20/IERC20.sol`
- `@openzeppelin/contracts/token/ERC20/SafeERC20.sol`
- `@openzeppelin/contracts/math/SafeMath.sol`
- `@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol`
- `@openzeppelin/contracts/cryptography/ECDSA.sol`
- `@openzeppelin/contracts/utils/ReentrancyGuard.sol`

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Boba Brewery
Platform	Other
Language	Solidity
Codebase	https://github.com/BobaBrewery/brewery-contarcts
Commit	81f55973fdf41025ffceb9acc31d451f8f3ad88a

Audit Summary

Delivery Date	Feb 24, 2022
Audit Methodology	Static Analysis, Manual Review

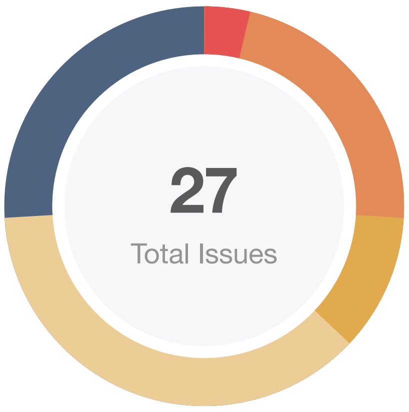
Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Mitigated	Resolved
● Critical	1	0	0	0	0	0	1
● Major	6	0	0	3	0	0	3
● Medium	3	0	0	0	0	0	3
● Minor	10	0	0	3	0	0	7
● Informational	7	0	0	1	0	0	6
● Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
IAC	contracts/interfaces/IAdmin.sol	8f11fa67577b373943127f3a87d25c5b661a370ac38b7494456e7af8ebc12462
IAS	contracts/interfaces/IAllocationStaking.sol	704e153b17fed66401db0a393756efae6633efc5541b5927cfad5c58f9bf2a68
IER	contracts/interfaces/IERC20Metadata.sol	9b69e478d33697ad1a9028694ea7dc5bd11da9039ef27ff07b940684efac6333
ISF	contracts/interfaces/ISalesFactory.sol	3e1e16f1307d8dc8b637d39eadd31a2737030144b7728f09cdd00df5e135fca6
SMC	contracts/math/SafeMath.sol	4a04d0a20a19e3ef1dcabae9cad9ba006430a4e7eec4d9b519db87999722c98a
BSC	contracts/sales/BrewerySale.sol	7801ac20c250306e1073affe8fa7097689635363c53f388b9ae73323d6ac5674
SFC	contracts/sales/SalesFactory.sol	b41b80e05eb03cf26c77e6e753d075a841ad848e9e6e23ad1543b196cdd813cf
CCK	contracts/utils/Context.sol	b13f7a4a8af9aa32577a61eaaccef4f8a9daeebc56f3e54a074c15b44df1217b
ACK	contracts/Admin.sol	380c5775228674a27a3d3422345745a3b4dfaf48ab9302ffdc9a166637853a74
ASC	contracts/AllocationStaking.sol	a23e81a1056c0b2f77ee92ec8e20534d1c3738d8ce3e100b4861aaa9d3471ca8
BTC	contracts/BreToken.sol	0ed96d53ce258304cd31695946d738b75f10e89266c434569765c8ed11d5ac70
IEC	contracts/IERC20.sol	91a59b940f265ebb6fd0426f3aea7ba14295a3a2a73f26c0434c35e84bf03513

Findings



🟥 **Critical**	**1** (3.70%)
🟧 **Major**	**6** (22.22%)
🟨 **Medium**	**3** (11.11%)
🟨 **Minor**	**10** (37.04%)
🟦 **Informational**	**7** (25.93%)
🟩 **Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Unlocked Compiler Version	Language Specific	● Informational	✓ Resolved
GLOBAL-02	Third Party Dependencies	Volatile Code	● Minor	ⓘ Acknowledged
GLOBAL-03	Missing Error Messages	Coding Style	● Informational	✓ Resolved
GLOBAL-04	Centralization Related Risks	**Centralization / Privilege**	● **Major**	ⓘ Acknowledged
ACK-01	Missing Input Validation for `_admins`	Volatile Code	● Medium	✓ Resolved
ASC-01	Missing Validation for `_pid`	Volatile Code	● Minor	✓ Resolved
ASC-02	Missing Zero Address Validation	Volatile Code	● Minor	✓ Resolved
ASC-03	Missing Validation for Same Pool	Volatile Code	● Medium	✓ Resolved
ASC-04	Incorrect `require` Condition	Logical Issue	● Informational	✓ Resolved
ASC-05	Wrong Design of Compound	Logical Issue	● Critical	✓ Resolved
ASC-06	`safeTransfer` Not Used	Volatile Code	● Minor	✓ Resolved
ASC-07	Incompatibility With Deflationary Tokens(Farming)	Volatile Code	● Minor	ⓘ Acknowledged
ASC-08	Potential Reentrancy Attack	Logical Issue	● Minor	✓ Resolved
ASC-09	SafeMath Not Used	Mathematical Operations	● Minor	✓ Resolved

ID	Title	Category	Severity	Status
ASC-10	Emergency Withdrawals Subject to Time Limit	Logical Issue	● Minor	ⓘ Acknowledged
BSC-01	Unused Base Contract `ReentrancyGuard`	Logical Issue	● Informational	✓ Resolved
BSC-02	Incorrect `require` Condition	Logical Issue	● Medium	✓ Resolved
BSC-03	Update Token Price Without Limit	**Centralization / Privilege**	● **Major**	ⓘ Acknowledged
BSC-04	Participating Missing `tokensDeposited` Check	Logical Issue	● Major	✓ Resolved
BSC-05	No Sold-out Check	Volatile Code	● Major	✓ Resolved
BSC-06	Missing Check `sale.tokensUnlockTime`	Volatile Code	● Major	✓ Resolved
BSC-07	Controversy about Participating Amount	Logical Issue	● Informational	ⓘ Acknowledged
BTC-01	Token Minted To Centralized Address	**Centralization / Privilege**	● **Major**	ⓘ Acknowledged
BTC-02	Missing Function `increaseAllowance`	Logical Issue	● Informational	✓ Resolved
SFC-01	Useless Variables and Event	Gas Optimization, Coding Style	● Informational	✓ Resolved
SFC-02	Missing Zero Address Validation	Volatile Code	● Minor	✓ Resolved
SFC-03	Incomplete Input Validation for `endIndex`	Volatile Code	● Minor	✓ Resolved

GLOBAL-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	Global	✓ Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

GLOBAL-02 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	● Minor	Global	ⓘ Acknowledged

Description

The contract is serving as the underlying entity to interact with third party protocols. The scope of the audit treats 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic of Boba Brewery requires interaction with `openzeppelin`, etc. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

From the client: as suggested, our team will constantly monitor the statuses of 3rd parties to mitigate the side effects.

GLOBAL-03 | Missing Error Messages

Category	Severity	Location	Status
Coding Style	● Informational	Global	✓ Resolved

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

In contract `AllocationStaking`, codes at Line 99 and 193 miss error messages.

In contract `Admin`, code at Line 49 miss an error message.

In contract `SalesFactory`, code at Line 36 miss an error message.

In contract `BrewerySale`, codes at Line 124, 141, 234, 244, 245, 502 miss error messages.

Recommendation

We advise refactoring the linked codes and provide string messages.

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

GLOBAL-04 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	**Global**	ⓘ Acknowledged

Description

In the contract `Admin`, the role `admin` has authority over the following functions:

- addAdmin
- removeAdmin

In the contract `SalesFactory`, the role `admin` has authority over the following functions:

- setAllocationStaking
- deploySale

In the contract `AllocationStaking`, the role `owner` has authority over the following functions:

- setSalesFactory
- add
- set

In the contract `BrewerySale`, the role `saleOwner` has authority over the following functions:

- depositTokens
- withdrawEarningsAndLeftover
- withdrawEarnings
- withdrawLeftover

the role `admin` has authority over the following functions:

- setVestingParams
- shiftVestingUnlockingTimes
- setSaleParams
- setSaleToken
- setRegistrationTime
- setSaleStart
- updateTokenPriceInETH
- postponeSale
- extendRegistrationPeriod

- setCap

Any compromise to these accounts may allow a hacker to take advantage of this authority.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

From the client: we have noticed the potential risk of centralization roles such as admin. We will be very careful to touch those roles and continue renouncing the ownership by time-lock and DAO in the long Term.

ACK-01 | Missing Input Validation For `_admins`

Category	Severity	Location	Status
Volatile Code	● Medium	contracts/Admin.sol: 19	✓ Resolved

Description

The given input `_admins` is missing the check for the non-zero address, and the array length should more than 0. If no admin being set in construct function, the contract `Admin` will be useless.

```
constructor (address [] memory _admins) public {
    for(uint i = 0; i < _admins.length; i++) {
        admins.push(_admins[i]);
        isAdmin[_admins[i]] = true;
    }
}
```

Recommendation

We advise adding the check for the passed-in values to prevent unexpected error as below:

```
constructor (address [] memory _admins) public {
    require(_admins.length > 0, "no admin address");
    for(uint i = 0; i < _admins.length; i++) {
        require(_admins[i] != address(0x0), "[RBAC] : Admin must be != than 0x0
address");
        admins.push(_admins[i]);
        isAdmin[_admins[i]] = true;
    }
}
```

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

ASC-01 | Missing Validation For `_pid`

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/AllocationStaking.sol: 137, 146, 152, 190, 200, 227, 255, 287, 308, 331	✓ Resolved

Description

In contract `AllocationStaking`, the given input `_pid` in the following functions needs to be checked if the index is valid.

- set()
- deposited()
- pending()
- setTokensUnlockTime()
- updatePool()
- deposit()
- withdraw()
- compound()
- emergencyWithdraw()
- getPendingAndDepositedForUsers()

Recommendation

We advise the client check as below:

```
require(_pid < poolInfo.length, "invalid _pid");
```

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

ASC-02 | Missing Zero Address Validation

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/AllocationStaking.sol: 82, 146, 152	✓ Resolved

Description

In contract `AllocationStaking`, the following functions miss the input validation. Address should be checked before assignment to make sure it is not zero addresses.

- initialize()
- deposited()
- pending()

Recommendation

Consider adding a zero check. As below:

```
require(salesFactory != address(0), "invalid address");
```

```
require(_user != address(0), "invalid address");
```

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

ASC-03 | Missing Validation For Same Pool

Category	Severity	Location	Status
Volatile Code	● Medium	contracts/AllocationStaking.sol: 117~118	✓ Resolved

Description

In contract `AllocationStaking`, when adding a new LP pool, the function `add` just add the new pool without checking whether the new LP exists. Only comment in the code to warn the owner is surely not enough.

```
// Add a new lp to the pool. Can only be called by the owner.
// DO NOT add the same LP token more than once. Rewards will be messed up if you do.
function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyOwner {
...
```

Recommendation

We recommend the client check whether the new LP pool exists before adding it in the function `add`.

Alleviation

The client changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

ASC-04 | Incorrect `require` Condition

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/AllocationStaking.sol: 291	✓ Resolved

Description

The related condition is not correct, since `user.amount` is always `>= 0`.

```
291  require(user.amount >= 0, "User does not have anything staked.");
```

Recommendation

We advise the client to change the condition as below:

```
291  require(user.amount > 0, "User does not have anything staked.");
```

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

ASC-05 | Wrong Design Of Compound

Category	Severity	Location	Status
Logical Issue	● Critical	contracts/AllocationStaking.sol: 287	✓ Resolved

Description

The design of compound is wrong. The staking tokens of different pools are different LP tokens while the reward token is an ERC20 token. The compound mechanism only can be used when the staking token and the reward token are the same.

If using the compound mechanism falsely, all the rewards and staking tokens amount will be messed up. Those who call the function `compound` can withdraw more LP tokens than their staking LP amount. The total LP supply of the pool is not increased, but their recorded LP amount is increased. When they withdraw more LP tokens, other users may have no LP token to withdraw.

Recommendation

We recommend remove the function `compound`.

Alleviation

The client added a require to make sure the function `compound` only can be called when the staking token is same as the reward token. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

ASC-06 | `safeTransfer` Not Used

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/AllocationStaking.sol: 326	✓ Resolved

Description

The function `erc20Transfer()` does not use the `safeTransfer()` from `SafeERC20`. If the function `transfer()` fails to transfer without reverting and just returns false, the function still goes on which will cause mistakes.

```
function erc20Transfer(address _to, uint256 _amount) internal {
    erc20.transfer(_to, _amount);
    paidOut += _amount;
}
```

Recommendation

We recommend refactoring function `erc20Transfer` as below, to prevent rounding error causing pool to not have enough reward token to pay.

```
function erc20Transfer(address _to, uint256 _amount) internal {
    uint256 balance = erc20.balanceOf(address(this));
    if(_amount > balance){
        erc20.safeTransfer(_to, balance);
        paidOut = paidOut.add(balance);
    }else{
        erc20.safeTransfer(_to, _amount);
        paidOut = paidOut.add(_amount);
    }
}
```

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

ASC-07 | Incompatibility With Deflationary Tokens(Farming)

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/AllocationStaking.sol	ⓘ Acknowledged

Description

When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user stakes 100 deflationary tokens (with a 10% transaction fee) in a MasterChef, only 90 tokens actually arrived in the contract. However, the user can still withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

The MasterChef takes the pool token balance(the `lpSupply`) into account when calculating the users' reward. An attacker can repeat the process of deposit and withdraw to lower the token balance(`lpSupply`) in a deflationary token pool and cause the contract to increase the reward amount.

Reference: https://thoreum-finance.medium.com/what-exploit-happened-today-for-gocerberus-and-garuda-also-for-lokum-ybear-piggy-caramelswap-3943ee23a39f

Recommendation

We advise the client to regulate the set of pool tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

From the client: we have been aware the potential risk and we will modify as suggested when we need to support deflationary tokens. For current token, we do not have buy-back or burn mechanism.

ASC-08 | Potential Reentrancy Attack

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/AllocationStaking.sol: 227, 255, 308	✓ Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

Since the `pool.lpToken` is unknown, the functions `deposit`, `withdraw` and `emergencyWithdraw` need a `ReentrancyGuard` to prevent potential reentrancy attack.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

ASC-09 | SafeMath Not Used

Category	Severity	Location	Status
Mathematical Operations	● Minor	contracts/AllocationStaking.sol: 112	✓ Resolved

Description

SafeMath from OpenZeppelin is not used in the following functions which makes them possible for overflow/underflow and will lead to an inaccurate calculation result.

- `fund()`
- `erc20Transfer()`

Recommendation

We advise the client to use OpenZeppelin's SafeMath library for all of the mathematical operations.

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

ASC-10 | Emergency Withdrawals Subject To Time Limit

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/AllocationStaking.sol: 311~312	ⓘ Acknowledged

Description

It is unusual that the emergency withdrawals are also subject to time limit. It is better to remove this time limit `user.tokensUnlockTime <= block.timestamp`.

```
// Withdraw without caring about rewards. EMERGENCY ONLY.
function emergencyWithdraw(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    require(user.tokensUnlockTime <= block.timestamp,
        "Emergency withdraw blocked during sale and cooldown period.");
...
```

Recommendation

We recommend removing the require `user.tokensUnlockTime <= block.timestamp`.

Alleviation

From the client:

Our project is a DAPP of Launchpad nature, and we have added restrictions in function `EmergencyWithdraw`, mainly to prevent following situations:

(1) To prevent individual users from injecting a large amount of token when registering for sale, and then withdrawing it directly after participating, so that they can maliciously grab a large share

(2) If a user registers for sale and then calls `EmergencyWithdraw` directly, the absence of this restriction will lead to confusion when we calculate the shares

(3) `EmergencyWithdraw` with this restriction can ensure that as long as a user registers in the project, his staking amount is constant in the project, ensuring fairness.

This feature should be almost impossible to use in practice. It is for the sake of logical integrity that we have added this `EmergencyWithdraw` function.

BSC-01 | Unused Base Contract `ReentrancyGuard`

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/sales/BrewerySale.sol: 12	✓ Resolved

Description

The contract `BrewerySale` has a base contract `ReentrancyGuard`, but never uses the modifier `nonReentrant` from `ReentrancyGuard`.

Recommendation

We recommend using the modifier `nonReentrant` in following functions.

- depositTokens
- withdrawTokens
- withdrawMultiplePortions
- withdrawEarningsAndLeftover
- withdrawEarnings
- withdrawLeftover

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

BSC-02 | Incorrect `require` Condition

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/sales/BrewerySale.sol: 331	✓ Resolved

Description

The related condition is not correct, since `sale.saleStart` has already added `timeToShift`.

```
323 function postponeSale(uint256 timeToShift) external onlyAdmin {
324     require(
325         block.timestamp < sale.saleStart,
326         "sale already started."
327     );
328     //  postpone registration start time
329     sale.saleStart = sale.saleStart.add(timeToShift);
330     require(
331         sale.saleStart + timeToShift < sale.saleEnd,
332         "Start time can not be greater than end time."
333     );
334 }
```

Recommendation

We advise the client to change the condition as below:

```
329 sale.saleStart = sale.saleStart.add(timeToShift);
330 require(
331     sale.saleStart < sale.saleEnd,
332     "Start time can not be greater than end time."
333 );
```

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

BSC-03 | Update Token Price Without Limit

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	contracts/sales/BrewerySale.sol: 315	ⓘ Acknowledged

Description

The owner can update token price to any price at any time. This may raise the community's concerns about the centralization issue.

Recommendation

We recommend the client to remove this function. To reduce reliance on the ETH volatility, it is better to use the price in USDT.

Alleviation

From the client: The initial token price is pegged to the ETH price in our current design. The method of update token price is to reduce reliance on the ETH volatility as suggested in the report. Our client would require a method to update the token price when a fix price token sale is conducted in our launchpad.

BSC-04 | Participating Missing `tokensDeposited` Check

Category	Severity	Location	Status
Logical Issue	● Major	contracts/sales/BrewerySale.sol: 382	✓ Resolved

Description

If the sale owner does not deposit tokens to the contract `BrewerySale`, and the buyers still call the function `participate` to buy the token, they can not withdraw any token finally. So it is important to ensure that the token has already been deposited in function `participate`.

Recommendation

We recommend the client to add a `require` in function `participate` as below:

```
require( sale.tokensDeposited, "Token has not been deposited");
```

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

BSC-05 | No Sold-out Check

Category	Severity	Location	Status
Volatile Code	● Major	contracts/sales/BrewerySale.sol: 434	✓ Resolved

Description

The function `participate` does not check if it is sold out or not when updating `sale.totalTokensSold`. The user may send the ETH and get no token.

Recommendation

We recommend the client to refactor code as below:

```
sale.totalTokensSold = sale.totalTokensSold.add(amountOfTokensBuying);
require( sale.totalTokensSold <= sale.amountOfTokensToSell, "token is sold out!");
```

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

BSC-06 | Missing Check `sale.tokensUnlockTime`

Category	Severity	Location	Status
Volatile Code	● Major	contracts/sales/BrewerySale.sol: 495	✓ Resolved

Description

The function `withdrawMultiplePortions` misses check for the `sale.tokensUnlockTime` like the function `withdrawTokens`.

Recommendation

We recommend the client to refactor as below:

```
495  function withdrawMultiplePortions(uint256 [] calldata portionIds) external {
496      require(block.timestamp >= sale.tokensUnlockTime, "Tokens can not be withdrawn
yet.");
497      ...
```

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

BSC-07 | Controversy About Participating Amount

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/sales/BrewerySale.sol: 403	ⓘ Acknowledged

Description

The input `amount` in function `participate` is signed by admin. Can you please tell us how you decide the participating amount of one user?

Recommendation

Please tell us how you decide the participating amount of one user.

Alleviation

From the client: the participating amount is roughly proportional to the amount of a user staked. We do have weighted factors and the minimum cap & maximum cap restrictions for each user to ensure a fair fundraising.

BTC-01 | Token Minted To Centralized Address

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	contracts/BreToken.sol: 26	ⓘ Acknowledged

Description

The amount of `totalSupply_` tokens that are minted to the contract deployer, may raise the community's concerns about the centralization issue.

Recommendation

We advise the client to carefully manage the deployer account's private key and avoid any potential risks of being hacked. We also advise the client to adopt Multisig, Timelock, and/or DAO in the project to manage this specific account in this case.

Alleviation

From the client: we will carefully manage the deployer account's private key (e.g. stored in a cold wallet) to avoid potential risks of being hacked.

BTC-02 | Missing Function `increaseAllowance`

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/BreToken.sol: 49	✓ Resolved

Description

The contract `BreToken` only has function `decreaseAllowance` but does not have function `increaseAllowance`. Usually these two functions come in pairs.

Recommendation

We recommend the client to add the function `increaseAllowance` as well.

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

SFC-01 | Useless Variables And Event

Category	Severity	Location	Status
Gas Optimization, Coding Style	● Informational	contracts/sales/SalesFactory.sol: 15~16, 22	✓ Resolved

Description

The variables `saleOwnerToSale` and `tokenToSale` are never used.

The event `SaleOwnerAndTokenSetInFactory` is never used.

Recommendation

We recommend the client to remove these useless codes or refactor the code and use they in a proper way.

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

SFC-02 | Missing Zero Address Validation

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/sales/SalesFactory.sol: 29~32	✓ Resolved

Description

In contract `SalesFactory`, the constructor misses the input validation. Address should be checked before assignment to make sure it is not zero addresses.

Recommendation

Consider adding a zero check. As below:

```
constructor (address _adminContract, address _allocationStaking) public {
    require(_adminContract != address(0), "invalid address");
    require(_allocationStaking != address(0), "invalid address");
    admin = IAdmin(_adminContract);
    allocationStaking = _allocationStaking;
}
```

Alleviation

The client added the input validation to `_adminContract`, but the `_allocationStaking` still misses input validation in the constructor since it can be zero address when deploying contract `SalesFactory`. The function `setAllocationStaking` has the zero address validation. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

SFC-03 | Incomplete Input Validation For `endIndex`

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/sales/SalesFactory.sol: 70	✓ Resolved

Description

Function `getAllSales` only checks the `endIndex > startIndex`, without checking `endIndex <= allSales.length`.

Recommendation

We advise adding the check for the passed-in values to prevent unexpected error as below:

```
require(endIndex > startIndex && endIndex <= allSales.length, "Bad input");
```

Alleviation

The client heeded our advice and changed related codes. Code change was applied in commit aa587f5d26d60fc11fc8db8e1a37166a73dd5f9e.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

CERTIK