CERTIK

Security Assessment

ShibaSwap

Jul 9th, 2021

Table of Contents

Summary

Overview

Project Summary

Audit Summary

Vulnerability Summary

Audit Scope

Findings

BBC-01 : Lack Of Input Validation

BBC-02 : Variable Declare as `Immutable`

BBD-01 : Centralization Risk

BLC-01 : Lack Of Input Validation

BLC-02 : Variable Declare as `Immutable`

BLK-01 : Potential Edge Case in Claimable Amount

BLK-02 : Centralization Risk

BLK-03 : Lack Of Input Validation

BSC-01 : Lack Of Input Validation

BSC-02 : Variable Declare as `Immutable`

BTC-01 : Delegation Should Move Along Fund Transfer

BTC-02 : Lack of Check for Integer Overflow

DBD-01 : Centralization Risk

DBD-02 : Lack of Event Emission for Significant Transactions

MDD-01 : Centralization Risk

MTL-01 : Centralization Risk

MTL-02 : Lack of Check for Integer Overflow

MTL-03 : Unrestricted Privilege Function

TCK-01 : Incorrect Reference URL In Comment

TDC-01 : add() Function Not Restricted

TDC-02 : Centralization Risk

TDC-03 : Over Minted Token

TDC-04 : Incompatibility With Deflationary Tokens

TDC-05 : Lack of Event Emission for Significant Transactions

TDC-06 : Misleading Result of Multiplier Calculation

TDC-07 : Inconsistent Checks-effects-interactions Pattern

TDC-08 : Potential Loss of Pool Rewards

TFC-01 : Centralization Risk
TFC-02 : Lack of Event Emission for Significant Transactions
TFC-03 : Potential Sandwich Attack
UVF-01 : Centralization Risk
UVF-02 : Reusable Code
UVF-03 : Lack of Event Emission for Significant Transactions
UVP-01 : Lack of Input Validation

Appendix

Disclaimer

About

Summary

This report has been prepared for Shiba to discover issues and vulnerabilities in the source code of the ShibaSwap 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.

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 given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	ShibaSwap
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/KaalDhairya/shibaswapv1/tree/SSwapv1-Certik
Commit	1305e7c127ea1d6dba78bd69aab367f53f8cd97e 9b182db842a581c1c793d40dce4d738ed14dcffb 51d237e488435b7f74588ccbb497b0d51aaf6764 949d75cc4bd8d23a0dc34ccb75f586ae01123cb6 6c6fed3662f811cfe95d3b49be730ce53c65fe95 58e2df72d15ed8e38074f98053d2281339d11169 22d2f0372a50a7d9e524b447ed9a91fc4e4212e6

Audit Summary

Delivery Date	Jul 09, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Partially Resolved	Resolved	Acknowledged	Declined
● Critical	0	0	0	0	0	0
● Major	8	0	0	8	0	0
● Medium	1	0	0	1	0	0
● Minor	11	0	1	10	0	0
● Informational	14	0	0	14	0	0
● Discussion	0	0	0	0	0	0

Audit Scope

ID	file	SHA256 Checksum

ShibaSwap Overview

The ShibaSwap Protocol is a part of Shiba token's decentralized ecosystem. It develops staking, distribution, and swapping features for the ecosystem.

The staking system is mainly implemented by the contracts:

- BoneToken.sol
- BuryBone.sol
- BuryLeash.sol
- BuryShib.sol

A new token, BONE, is introduced in the system. Users can deposit their BONE/LEASH/SHIB tokens to these contracts and get corresponding tBONE/xLEASH/xSHIB.

The distribution system is mainly implemented by the contracts:

- BoneLocker.sol
- DevBoneDistributor.sol
- MultiTokenLocker.sol
- TopDog.sol
- merkleDistributors/XXXMerkleDistributor.sol

Some of the rewards will be sent to the locking contract `BoneLocker`. These rewards will not be withdrawable until reaching the end of the locking period. Other rewards will be distributed directly to dev and user accounts.

The swapping system is mainly implemented by the contracts:

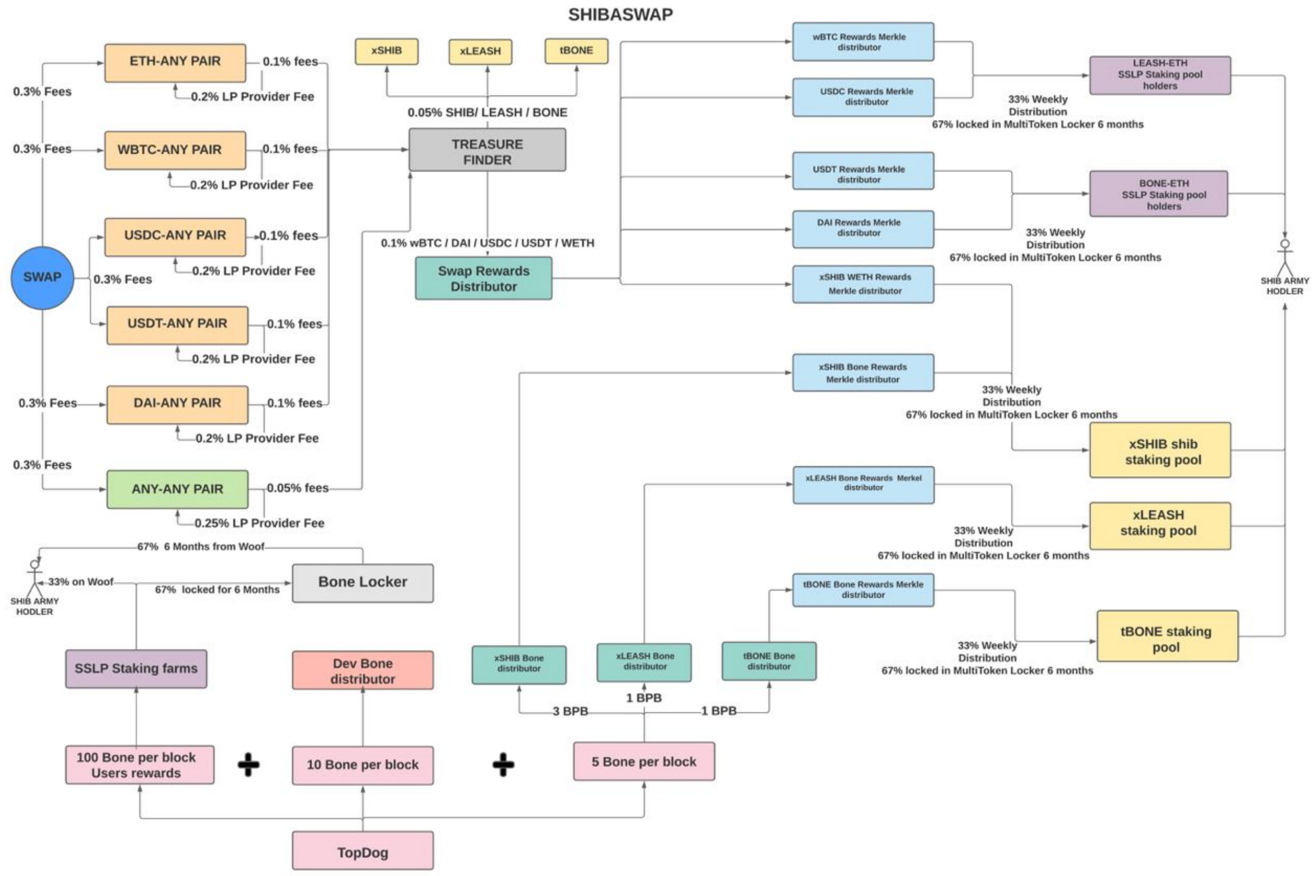
- Migrator.sol
- ShibaSushiFetch.sol
- ShibaUniFetch.sol
- TreasureFinder.sol
- uniswapv2/UniswapV2ERC20.sol
- uniswapv2/UniswapV2Factory.sol

- uniswapv2/UniswapV2Pair.sol
- uniswapv2/UniswapV2Router02.sol

This system allows users to migrate their LP tokens from their original pairing pools to the pairing pools provided by Shiba. In addition, it allows users to perform regular operations in pairing pools, such as adding/removing liquidity and swapping between different assets.

Shibaswap Architecture & Fee Models

ShibaSwap, a decentralized cryptocurrency exchange where users can exchange tokens. The diagram below illustrates how the unique flagship tokens, fee model, and incentivized mechanism.



Contract Dependencies

In ShibaSwap, the system inherits or uses a few of the depending injection contracts or addresses to fulfill the need of its complex business logic.

- `bone` for the contract `BasicBoneDistributor`;
- `boneToken` for the contract `BoneLocker`;
- `bone` for the contract `BuryBone`;
- `LEASH` for the contract `BuryLeash`;
- `shib` for the contract `BuryShib`;
- `bone` for the contract `DevBoneDistributor`;
- `chef`, `oldFactory`, `factory` for the contract `Migrator`;
- `oldRouter` and `router` for the contract `ShibaSushiFetch`;
- `oldRouter` and `router` for the contract `ShibaSushiFetch`;
- `bone`, `boneLocker`, `migrator` and `poolInfo[].lpToken` for the contract `TopDog`;
- `factory`, `bone`, `shiba`, `leash` and all other tokens used in swappings for the contract `TreasureFinder`;
- `token` for the contract `boneMerkleDistributor`;
- `token` for the contract `daiMerkleDistributor`;
- `token` for the contract `usdcMerkleDistributor`;
- `token` for the contract `usdtMerkleDistributor`;
- `token` for the contract `wbtcMerkleDistributor`;
- `token` for the contract `wethMerkleDistributor`;
- `token` for the contract `xLeashBoneMerkleDistributor`;
- `token` for the contract `xShibBoneMerkleDistributor`;
- `factory`, `token0` and `token1` for the contract `UniswapV2Pair`;
- `factory` and `WETH` for the contract `UniswapV2Router02`.

We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

Privileged Roles

To set up the project correctly, improve overall project quality and preserve upgradability, the following roles are adopted in the codebase:

- `owner` is adopted to withdraw bone in the contract `BasicBoneDistributor`;
- `owner` is adopted to lock token and withdraw all token in the contract `BoneLocker`;
- `owner` is adopted to the mint token in the contract `BoneToken`;
- `owner` is adopted to set wallet addresses and percentage of distributions in the contract `DevBoneDistributor`;
- `owner` is adopted to withdraw tokens in the contract `SwapRewardDistributor`;
- `admin` is adopted to queue, cancel and execute transactions in the contract `Timelock`;
- `owner` is adopted to set up a new pool, update pool configurations, change token distributors, modify distribution percentages, update locking period and withdraw tokens from the locker in the contract `TopDog`.
- `owner` is adopted to update Merkle root and withdraw tokens in the contract `boneMerkleDistributor`;
- `owner` is adopted to freeze/unfreeze the contract, update Merkle root and withdraw tokens in the contract `daiMerkleDistributor`;
- `owner` is adopted to freeze/unfreeze the contract, update Merkle root and withdraw tokens in the contract `usdcMerkleDistributor`;
- `owner` is adopted to freeze/unfreeze the contract, update Merkle root and withdraw tokens in the contract `usdtMerkleDistributor`;
- `owner` is adopted to freeze/unfreeze the contract, update Merkle root and withdraw tokens in the contract `wbtcMerkleDistributor`;
- `owner` is adopted to freeze/unfreeze the contract, update Merkle root and withdraw tokens in the contract `wethMerkleDistributor`;
- `owner` is adopted to freeze/unfreeze the contract, update Merkle root and withdraw tokens in the contract `xLeashBoneMerkleDistributor`;
- `owner` is adopted to freeze/unfreeze the contract, update Merkle root and withdraw tokens in the contract `xShibBoneMerkleDistributor`;
- `feeToSetter` is adopted to set fee recipient, migratory, and fees in the contract `UniswapV2Factory`.

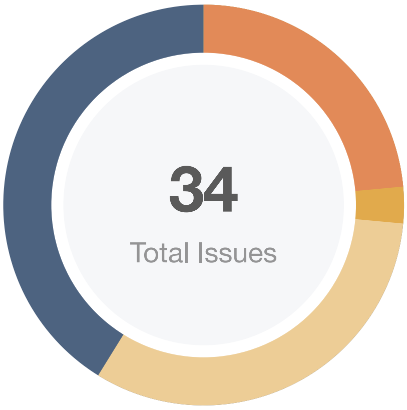
To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Furthermore, any plan to invoke the aforementioned functions should also be considered to move to the execution queue of the `Timelock` contract.

According to the Shiba Inu Ecosystem Woof Paper, Multisig wallets will be used for privileged roles. The addresses are listed as follows:

- MULTISIG ADDRESS: 0x38e1d4314a38c60C6ab3b98b0a89a4411D839d44
    - @OMEGA_HYPERION: 0x399EC033EE08241512212a4C388a76C9d3aB1c00
    - @KAAL_DHAIRYA: 0xBab4F3e701F6d2e009Af3C7f1eF2e7dD68225E96
    - @HYROSHI_KIPA: 0x80e32DEfc16ce8f78d09E6ef7065AfE031bAcab7
    - @JUNE_HORLA: 0x6948cBbEa74549062050a164d8fc4cFF27E82084
    - @SISLEY_ARGONAUT: 0xe166c948b8aED157575B6792019cdeE8a5177dcE
- MULTISIG EMERGENCY ADDRESS: 0x4267A3aD7d20c2396ebb0Fe72119984F7073761C
    - @OMEGA_HYPERION: 0x399EC033EE08241512212a4C388a76C9d3aB1c00
    - @KAAL_DHAIRYA: 0xBab4F3e701F6d2e009Af3C7f1eF2e7dD68225E96
    - @HYROSHI_KIPA: 0x80e32DEfc16ce8f78d09E6ef7065AfE031bAcab7
    - @JUNE_HORLA: 0x6948cBbEa74549062050a164d8fc4cFF27E82084
    - @SISLEY_ARGONAUT: 0xe166c948b8aED157575B6792019cdeE8a5177dcE
    - @COUNTER_NOMAD: 0x8E1B6Af660C14f5CC28727f23fCcBC977bd89B6B
    - @SHINATO_SAMA: 0x6b162Bc637bAAe0DAC38c200D9727fc679a0cCE4
    - @MISS_PHOENIX_SHIB: 0x30f45F7b08164D2Dd38D9Cdd8509b1E580432d04
    - @BURF_DURF: 0x5D471E3a033EaF7eE0cA303405978Da4c2cdAD33

Multisig, which is used for `devWallet`, requires 3 out of 5 signatures for a transaction to be approved. Emergency Multisig, which is used for all other privileged roles, requires 6 out of 9 signatures for a transaction to be approved.

Findings



🟥 **Critical**	**0** (0.00%)
🟧 **Major**	**8** (23.53%)
🟨 **Medium**	**1** (2.94%)
🟨 **Minor**	**11** (32.35%)
🟦 **Informational**	**14** (41.18%)
🟩 **Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
BBC-01	Lack Of Input Validation	Volatile Code	● Informational	✓ Resolved
BBC-02	Variable Declare as `Immutable`	Gas Optimization	● Informational	✓ Resolved
BBD-01	Centralization Risk	**Centralization / Privilege**	● **Minor**	✓ **Resolved**
BLC-01	Lack Of Input Validation	Volatile Code	● Informational	✓ Resolved
BLC-02	Variable Declare as `Immutable`	Gas Optimization	● Informational	✓ Resolved
BLK-01	Potential Edge Case in Claimable Amount	Logical Issue	● Minor	✓ Resolved
BLK-02	Centralization Risk	**Centralization / Privilege**	● **Major**	✓ **Resolved**
BLK-03	Lack Of Input Validation	Volatile Code	● Informational	✓ Resolved
BSC-01	Lack Of Input Validation	Volatile Code	● Informational	✓ Resolved
BSC-02	Variable Declare as `Immutable`	Gas Optimization	● Informational	✓ Resolved
BTC-01	Delegation Should Move Along Fund Transfer	Logical Issue	● Major	✓ Resolved
BTC-02	Lack of Check for Integer Overflow	Mathematical Operations	● Informational	✓ Resolved
DBD-01	Centralization Risk	**Centralization / Privilege**	● **Minor**	✓ **Resolved**

ID	Title	Category	Severity	Status
DBD-02	Lack of Event Emission for Significant Transactions	Coding Style	Informational	Resolved
MDD-01	Centralization Risk	**Centralization / Privilege**	**Major**	**Resolved**
MTL-01	Centralization Risk	**Centralization / Privilege**	**Major**	**Resolved**
MTL-02	Lack of Check for Integer Overflow	Mathematical Operations	Minor	Resolved
MTL-03	Unrestricted Privilege Function	Logical Issue	Medium	Resolved
TCK-01	Incorrect Reference URL In Comment	Coding Style	Informational	Resolved
TDC-01	add() Function Not Restricted	Volatile Code	Major	Resolved
TDC-02	Centralization Risk	**Centralization / Privilege**	**Minor**	**Resolved**
TDC-03	Over Minted Token	Logical Issue	Minor	Resolved
TDC-04	Incompatibility With Deflationary Tokens	Logical Issue	Minor	Resolved
TDC-05	Lack of Event Emission for Significant Transactions	Coding Style	Informational	Resolved
TDC-06	Misleading Result of Multiplier Calculation	Logical Issue	Minor	Resolved
TDC-07	Inconsistent Checks-effects-interactions Pattern	Logical Issue	Major	Resolved
TDC-08	Potential Loss of Pool Rewards	Logical Issue	Minor	Resolved
TFC-01	Centralization Risk	**Centralization / Privilege**	**Minor**	**Resolved**
TFC-02	Lack of Event Emission for Significant Transactions	Coding Style	Informational	Resolved
TFC-03	Potential Sandwich Attack	Logical Issue	Minor	Partially Resolved
UVF-01	Centralization Risk	**Centralization / Privilege**	**Major**	**Resolved**

ID	Title	Category	Severity	Status
UVF-02	Reusable Code	Gas Optimization	● Informational	✓ Resolved
UVF-03	Lack of Event Emission for Significant Transactions	Coding Style	● Informational	✓ Resolved
UVP-01	Lack of Input Validation	Volatile Code	● Major	✓ Resolved

BBC-01 | Lack Of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	projects/shibaswapv1/contracts/BuryBone.sol: 18	✓ Resolved

Description

In the contract `BuryBone`, the given constructor input `_bone` is missing a sanity check for ensuring a non-zero address will assign.

Recommendation

We recommend adding check for the passed-in value is non-zero to prevent any unexpected error. Example:

```
require(address(_bone) != address(0), "_bone is a zero address");
```

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

BBC-02 | Variable Declare as `Immutable`

Category	Severity	Location	Status
Gas Optimization	● Informational	projects/shibaswapv1/contracts/BuryBone.sol: 15	✓ Resolved

Description

The variable `bone` assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since `immutable` will not be stored in storage. Still, values will directly insert the values into the runtime code.

Recommendation

We recommend using an immutable state variable for `bone`.

```
15  IERC20 public immutable bone;
```

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

BBD-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Minor	projects/shibaswapv1/contracts/BasicBoneDistributor.sol: 20~21	✓ Resolved

Description

In the contract `bone`, the role `owner` has authority over the following function:

- `withdrawBone()`: withdraw the ERC20 token `bone` with the arbitrary amount to any `_destination` address.

Any compromise to the account `owner` may allow the hacker to take advantage of it and transfer the withdrawn tokens to an arbitrary address, the `_destination` address.

As `BasicBoneDistributor` is an abstract contract, it is highly recommended to follow best practices by managing and interacting with any contract inheriting from `BasicBoneDistributor` through a decentralized mechanism.

For example:

- `tBoneBoneDistributor()`
- `xLeashBoneDistributor()`
- `xShibBoneDistributor()`

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly encourage the centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Shiba**]: The team acknowledges the issue and applied the MultiSig solution in any sensitive privilege access(i.e., `owner` role).

The Emergency Multisig members have trusted members of the Community and the Defi environment. There must be 6 out of 9 signatures from the below addresses for a transaction to be approved.

- MULTISIG EMERGENCY ADDRESS: 0x4267A3aD7d20c2396ebb0Fe72119984F7073761C
    - @OMEGA_HYPERION: 0x399EC033EE08241512212a4C388a76C9d3aB1c00
    - @KAAL_DHAIRYA: 0xBab4F3e701F6d2e009Af3C7f1eF2e7dD68225E96
    - @HYROSHI_KIPA: 0x80e32DEfc16ce8f78d09E6ef7065AfE031bAcab7
    - @JUNE_HORLA: 0x6948cBbEa74549062050a164d8fc4cFF27E82084
    - @SISLEY_ARGONAUT: 0xe166c948b8aED157575B6792019cdeE8a5177dcE
    - @COUNTER_NOMAD: 0x8E1B6Af660C14f5CC28727f23fCcBC977bd89B6B
    - @SHINATO_SAMA: 0x6b162Bc637bAAe0DAC38c200D9727fc679a0cCE4
    - @MISS_PHOENIX_SHIB: 0x30f45F7b08164D2Dd38D9Cdd8509b1E580432d04
    - @BURF_DURF: 0x5D471E3a033EaF7eE0cA303405978Da4c2cdAD33

You can find more details about the MultiSig Model include the settings and member's information in the the Shiba Inu Ecosystem Woof Paper Page 24.

BLC-01 | Lack Of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	projects/shibaswapv1/contracts/BuryLeash.sol: 18	✓ Resolved

Description

The given input `LEASH` is missing the sanity check for the non-zero address in the aforementioned line.

Recommendation

We recommend adding check for the passed-in value is non-zero to prevent any unexpected error.

Example:

```
    require(address(_LEASH) != address(0), "_LEASH is a zero address");
```

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

BLC-02 | Variable Declare as `Immutable`

Category	Severity	Location	Status
Gas Optimization	● Informational	projects/shibaswapv1/contracts/BuryLeash.sol: 15	✓ Resolved

Description

The variable `LEASH` assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since `immutable` will not be stored in storage. Still, values will directly insert the values into the runtime code.

Recommendation

We recommend using an immutable state variable for `LEASH`.

```
15  IERC20 public immutable LEASH;
```

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

BLK-01 | Potential Edge Case in Claimable Amount

Category	Severity	Location	Status
Logical Issue	● Minor	projects/shibaswapv1/contracts/BoneLocker.sol: 51~55, 73~76	✓ Resolved

Description

In the aforementioned lines, the claimable amount will be affected when the address's role changes (`lockInfoByUser[account][i]._isDev`). Therefore, the claimable amount could be different from the token amount at their unlocking time.

```
function getClaimableAmount(address _user) public view returns(uint256) {
    LockInfo[] memory lockInfoArrayForUser = lockInfoByUser[_user];
    ...
    uint256 lockingPeriodHere = lockingPeriod;
    if(lockInfoArrayForUser[i]._isDev){
        lockingPeriodHere = devLockingPeriod;
    }
    for (i; i<lockInfoArrayForUser.length; i++){
        if(now >= (lockInfoArrayForUser[i]._timestamp.add(lockingPeriodHere))){
           totalTransferableAmount =
totalTransferableAmount.add(lockInfoArrayForUser[i]._amount);
        }
    ...
    }
    return totalTransferableAmount;
}
```

The following is a potential scenario. Assume that the `lockingPeriod` is 10 days while `devLockingPeriod` is 1 day:

- Day 1: A non-dev account receives some locked tokens and expects to unlock them on Day 11. Lock info is stored at `lockInfoByUser[account][0]`.
- Day 2: The account is set as a dev account. It receives some locked tokens and expects to unlock them on Day 3. Lock info is stored at `lockInfoByUser[account][1]`.
- Day 3: The account should be able to unlock tokens received on Day 2. However, when it calls `claimAll()`, it is still not able to claim these tokens because `now < (lockInfoByUser[account][0]._timestamp.add(lockingPeriod))` (3 < 1 + 10).

Recommendation

We would like to confirm if the above-mentioned case could be a potential edge case in the real-world scenario.

Alleviation

[Shiba]: The team acknowledged the finding and disagreed on it. The Shiba team confirmed that the `BoneLocker` contract's owner is `TopDog`, where the `boneLocker.lock()` function is called in two situations:

- for users making a deposit/withdraw (basically harvest);
- for the `devBoneDistributor` address, which is a contract, when `updatePool` is triggered.

The team will ensure never make the dev address (`devBoneDistributor`) as any user address, and it will always be the `devBoneDistributor` smart contract address. There will be no such case where any address will be a normal address, then set as a dev address and then gets it token locked as a dev address for less `devLockingPeriod` than `lockingPeriod`. Therefore, an address will either be a dev address or will not be a dev address.

[CertiK]: We agreed that the issue wouldn't occur if an address will either be a dev address or will not be a dev address.

BLK-02 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Major	projects/shibaswapv1/contracts/BoneLocker.sol: 107	✓ Resolved

Description

In the contract `BoneLocker`, the role `owner` has the authority over the following function:

- `emergencyWithdrawOwner()`: withdraw all the ERC20 token `boneToken` to any arbitrary address `_to`.

Any compromise to the `owner` account may allow the hacker to take advantage of this and transfer the withdrawn tokens to an arbitrary address, which is the `_to` address.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Shiba**]: The team acknowledges the issue and applied the MultiSig solution in any sensitive privilege access(i.e., `owner` role).

The Emergency Multisig members have trusted members of the Community and the Defi environment. There must be 6 out of 9 signatures from the below addresses for a transaction to be approved.

- MULTISIG EMERGENCY ADDRESS: 0x4267A3aD7d20c2396ebb0Fe72119984F7073761C
    - @OMEGA_HYPERION: 0x399EC033EE08241512212a4C388a76C9d3aB1c00
    - @KAAL_DHAIRYA: 0xBab4F3e701F6d2e009Af3C7f1eF2e7dD68225E96
    - @HYROSHI_KIPA: 0x80e32DEfc16ce8f78d09E6ef7065AfE031bAcab7
    - @JUNE_HORLA: 0x6948cBbEa74549062050a164d8fc4cFF27E82084

- @SISLEY_ARGONAUT: 0xe166c948b8aED157575B6792019cdeE8a5177dcE
- @COUNTER_NOMAD: 0x8E1B6Af660C14f5CC28727f23fCcBC977bd89B6B
- @SHINATO_SAMA: 0x6b162Bc637bAAe0DAC38c200D9727fc679a0cCE4
- @MISS_PHOENIX_SHIB: 0x30f45F7b08164D2Dd38D9Cdd8509b1E580432d04
- @BURF_DURF: 0x5D471E3a033EaF7eE0cA303405978Da4c2cdAD33

You can find more details about the MultiSig Model include the settings and member's information in the the Shiba Inu Ecosystem Woof Paper Page 24.

BLK-03 | Lack Of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	projects/shibaswapv1/contracts/BoneLocker.sol: 113	✓ Resolved

Description

The given input `emergencyAddress` is missing the sanity check for the non-zero address in the aforementioned line.

Recommendation

We recommend adding a check that the passed-in value is non-zero to prevent unexpected behavior. Example:

```
    require(_newAddr != address(0), "_newAddr is a zero address");
```

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

BSC-01 | Lack Of Input Validation

Category	Severity	Location	Status
Volatile Code	● Informational	projects/shibaswapv1/contracts/BuryShib.sol: 18	✓ Resolved

Description

The given input `_shib` is missing the sanity check for the non-zero address in the aforementioned line.

Recommendation

We recommend adding the check for the passed-in values is non-zero to prevent unexpected error.

Example:

```
    require(address(_shib) != address(0), "_shib is a zero address");
```

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

BSC-02 | Variable Declare as `Immutable`

Category	Severity	Location	Status
Gas Optimization	● Informational	projects/shibaswapv1/contracts/BuryShib.sol: 15	✓ Resolved

Description

The variable `shib` assigned in the constructor can declare as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since `immutable` will not be stored in storage. Still, values will directly insert the values into the runtime code.

Recommendation

We recommend using an immutable state variable for `shib`.

```
15  IERC20 public immutable shib;
```

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

BTC-01 | Delegation Should Move Along Fund Transfer

Category	Severity	Location	Status
Logical Issue	● Major	projects/shibaswapv1/contracts/BoneToken.sol: 15	✓ Resolved

Description

Given `BoneToken` is a governance token, any functions that involve the fund operation, such as transfer/mint/burn, should also require come along with the delegate operation. Otherwise, it could lead to an inconsistency in the result of the delegate of each addresses.

For example:

- `transfer()`
- `transferFrom()`
- `burn()`

Recommendation

We advise that `transfer()`, `transferFrom()` and `burn()` functions are properly overridden to also transfer delegates on each invocation from the sender of the funds to the recipient.

Alleviation

[**Shiba**]: The team heeded our advice and resolved this issue in the commit `58e2df72d15ed8e38074f98053d2281339d11169`.

BTC-02 | Lack of Check for Integer Overflow

Category	Severity	Location	Status
Mathematical Operations	● Informational	projects/shibaswapv1/contracts/BoneToken.sol: 232	✓ Resolved

Description

The operation in the aforementioned line does not check integer overflow:

```
232        numCheckpoints[delegatee] = nCheckpoints + 1;
```

It might lead to an inaccurate result.

Recommendation

We advise the client to check integer overflows in integer operations.

Alleviation

[**Shiba**]: The team heeded our advice and resolved this issue in the commit `58e2df72d15ed8e38074f98053d2281339d11169`.

DBD-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Minor	projects/shibaswapv1/contracts/DevBoneDistributor.sol: 41, 45, 49, 53	✓ Resolved

Description

The owner of the contract with the `owner` role has the privilege to update the following sensitive variables:

- `devWallet`
- `marketingWallet`
- `adminWallet`
- `devSharePercent`
- `marketingSharePercent`
- `adminSharePercent`

All of these variables decide the source and the percentage of BONE that will be distributed to `devWallet` and `marketingAndGrowthWallet`. Any compromise to the `owner` account may allow the hacker to take advantage of it and potentially transfer all BONE tokens to any arbitrary address.

Recommendation

We recommend the team carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Shiba**]: The team acknowledges the issue and applied the MultiSig solution in any sensitive privilege access(i.e., `owner` role).

The Emergency Multisig members have trusted members of the Community and the Defi environment. There must be 6 out of 9 signatures from the below addresses for a transaction to be approved.

- MULTISIG EMERGENCY ADDRESS: 0x4267A3aD7d20c2396ebb0Fe72119984F7073761C
    - @OMEGA_HYPERION: 0x399EC033EE08241512212a4C388a76C9d3aB1c00
    - @KAAL_DHAIRYA: 0xBab4F3e701F6d2e009Af3C7f1eF2e7dD68225E96
    - @HYROSHI_KIPA: 0x80e32DEfc16ce8f78d09E6ef7065AfE031bAcab7
    - @JUNE_HORLA: 0x6948cBbEa74549062050a164d8fc4cFF27E82084
    - @SISLEY_ARGONAUT: 0xe166c948b8aED157575B6792019cdeE8a5177dcE
    - @COUNTER_NOMAD: 0x8E1B6Af660C14f5CC28727f23fCcBC977bd89B6B
    - @SHINATO_SAMA: 0x6b162Bc637bAAe0DAC38c200D9727fc679a0cCE4
    - @MISS_PHOENIX_SHIB: 0x30f45F7b08164D2Dd38D9Cdd8509b1E580432d04
    - @BURF_DURF: 0x5D471E3a033EaF7eE0cA303405978Da4c2cdAD33

You can find more details about the MultiSig Model include the settings and member's information in the the Shiba Inu Ecosystem Woof Paper Page 24.

DBD-02 | Lack of Event Emission for Significant Transactions

Category	Severity	Location	Status
Coding Style	● Informational	projects/shibaswapv1/contracts/DevBoneDistributor.sol: 41, 45, 49, 53	✓ Resolved

Description

Functions that affect the status of sensitive variables should be able to emit events as notifications to customers:

- `setDevWallet()`
- `setMarketingWallet()`
- `setAdminWallet()`
- `setWalletDistribution()`

Recommendation

We advise the client to consider adding events for sensitive actions and emit them in the corresponding functions.

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

The team removed `adminWallet` and merged it into `devWallet` in the commit `22d2f0372a50a7d9e524b447ed9a91fc4e4212e6`. Events are modified and emitted in the updated functions.

MDD-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Major	projects/shibaswapv1/contracts/merkleDistributors/boneMerkleDistributor.sol: 261, 267, 284	✓ Resolved

Description

The owner of the contract with the `owner` role has the privilege to update the `merkleRoot` by calling function `updateMerkleRoot()`. Any compromise to the account with `owner` role may allow the hacker to take advantage of it. For example, if a hacker passes the argument `merkleProof` when calling function `claim()`, they bypass the check `require(MerkleProof.verify(merkleProof, merkleRoot, node), 'MerkleDistributor: Invalid proof.')` in L267. Because of this manipulation of `merkleRoot`, they could transfer any `amount` of `token` to an arbitrary address `account`.

The same concern exists in all of these contracts as the contract `boneMerkleDistributor` has almost exactly the same content as:

- `daiMerkleDistributor`
- `usdcMerkleDistributor`
- `usdtMerkleDistributor`
- `wbtcMerkleDistributor`
- `wethMerkleDistributor`
- `xLeashBoneMerkleDistributor`
- `xShibBoneMerkleDistributor`

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked.
In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;

- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Shiba**]: The team acknowledges the issue and applied the MultiSig solution in any sensitive privilege access(i.e., `owner` role).

The Emergency Multisig members have trusted members of the Community and the Defi environment. There must be 6 out of 9 signatures from the below addresses for a transaction to be approved.

- MULTISIG EMERGENCY ADDRESS: 0x4267A3aD7d20c2396ebb0Fe72119984F7073761C
    - @OMEGA_HYPERION: 0x399EC033EE08241512212a4C388a76C9d3aB1c00
    - @KAAL_DHAIRYA: 0xBab4F3e701F6d2e009Af3C7f1eF2e7dD68225E96
    - @HYROSHI_KIPA: 0x80e32DEfc16ce8f78d09E6ef7065AfE031bAcab7
    - @JUNE_HORLA: 0x6948cBbEa74549062050a164d8fc4cFF27E82084
    - @SISLEY_ARGONAUT: 0xe166c948b8aED157575B6792019cdeE8a5177dcE
    - @COUNTER_NOMAD: 0x8E1B6Af660C14f5CC28727f23fCcBC977bd89B6B
    - @SHINATO_SAMA: 0x6b162Bc637bAAe0DAC38c200D9727fc679a0cCE4
    - @MISS_PHOENIX_SHIB: 0x30f45F7b08164D2Dd38D9Cdd8509b1E580432d04
    - @BURF_DURF: 0x5D471E3a033EaF7eE0cA303405978Da4c2cdAD33

You can find more details about the MultiSig Model include the settings and member's information in the the Shiba Inu Ecosystem Woof Paper Page 24.

MTL-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Major	projects/shibaswapv1/contracts/MultiTokenLocker.sol: 41, 56	✓ Resolved

Description

In the contract `MultiTokenLocker`, the role `owner` has authority over the following function:

- `withdrawTheseToken()`: transfer a list of unlocked tokens to a list of accounts.
- `withdrawThisToken()`: transfer an unlocked token to an account.

These two functions have the possibility of being maliciously manipulated by hackers if the account of the `owner` was compromised.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly encourage the centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Shiba**]: The team acknowledges the issue and applied the MultiSig solution in any sensitive privilege access(i.e., `owner` role).

The Emergency Multisig members have trusted members of the Community and the Defi environment. There must be 6 out of 9 signatures from the below addresses for a transaction to be approved.

- MULTISIG EMERGENCY ADDRESS: 0x4267A3aD7d20c2396ebb0Fe72119984F7073761C
    - @OMEGA_HYPERION: 0x399EC033EE08241512212a4C388a76C9d3aB1c00
    - @KAAL_DHAIRYA: 0xBab4F3e701F6d2e009Af3C7f1eF2e7dD68225E96
    - @HYROSHI_KIPA: 0x80e32DEfc16ce8f78d09E6ef7065AfE031bAcab7

  - @JUNE_HORLA: 0x6948cBbEa74549062050a164d8fc4cFF27E82084
  - @SISLEY_ARGONAUT: 0xe166c948b8aED157575B6792019cdeE8a5177dcE
  - @COUNTER_NOMAD: 0x8E1B6Af660C14f5CC28727f23fCcBC977bd89B6B
  - @SHINATO_SAMA: 0x6b162Bc637bAAe0DAC38c200D9727fc679a0cCE4
  - @MISS_PHOENIX_SHIB: 0x30f45F7b08164D2Dd38D9Cdd8509b1E580432d04
  - @BURF_DURF: 0x5D471E3a033EaF7eE0cA303405978Da4c2cdAD33

You can find more details about the MultiSig Model include the settings and members' information in the the Shiba Inu Ecosystem Woof Paper Page 24.

MTL-02 | Lack of Check for Integer Overflow

Category	Severity	Location	Status
Mathematical Operations	● Minor	projects/shibaswapv1/contracts/MultiTokenLocker.sol: 59	Resolved

Description

The operation in the aforementioned line does not check integer overflow:

```
     require(block.timestamp >= lockInfoArray[_lockId]._timestamp +
lockInfoArray[_lockId]._lockingPeriod, "Cannot claim now, still in locking period");
```

It might lead to an inaccurate result.

Recommendation

We advise the client to consider using `SafeMath` library of Openzeppelin library:

```
    require(block.timestamp >=
lockInfoArray[_lockId]._timestamp.add(lockInfoArray[_lockId]._lockingPeriod), "Cannot
claim now, still in locking period");
```

Alleviation

[**Shiba**]: The team heeded our advice and resolved this issue in the commit `6c6fed3662f811cfe95d3b49be730ce53c65fe95`.

MTL-03 | Unrestricted Privilege Function

Category	Severity	Location	Status
Logical Issue	● Medium	projects/shibaswapv1/contracts/MultiTokenLocker.sol: 31	✓ Resolved

Description

The function `MultiTokenLocker.receiveApproval()` transfers tokens from `_distributorContract` to the contract account. It is not restricted, so everyone can call this function. Its safety is guaranteed by the fact that `_distributorContract` needs to approve some allowance for this contract before this function is triggered, or `_distributorContract` does not hold any token until it triggers this function. However, the logic in `_distributorContract` before calling the function `MultiTokenLocker.receiveApproval()` is unknown to us, which means the safety of this function is not guaranteed.

Recommendation

We advise the client add restrictions on calling the function `MultiTokenLocker.receiveApproval()`, `onlyOwner` as an example, or review the design of `_distributorContract` to ensure `MultiTokenLocker.receiveApproval()` will not be triggered only when it is necessary.

Alleviation

[**Shiba**]: The team heeded our advice and resolved this issue in the commit `6c6fed3662f811cfe95d3b49be730ce53c65fe95`.

TCK-01 | Incorrect Reference URL In Comment

Category	Severity	Location	Status
Coding Style	● Informational	projects/shibaswapv1/contracts/Timelock.sol: 3	✓ Resolved

Description

In the aforementioned line, the comment of reference URL to timelock contract is incorrect.

Recommendation

We recommend addressing the comment to correct reference URL to `https://raw.githubusercontent.com/compound-finance/compound-protocol/master/contracts/Timelock.sol`

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

TDC-01 | add() Function Not Restricted

Category	Severity	Location	Status
Volatile Code	● Major	projects/shibaswapv1/contracts/TopDog.sol: 139	✓ Resolved

Description

When the same LP token is added into a pool more than once in function `add()`, the total amount of reward in function `updatePool()` will be incorrectly calculated. The current implementation is relying on the operation correctness to avoid repeatedly adding the same LP token to the pool, as the function will only be called by the owner.

Recommendation

We recommend adding the check for ensuring whether the given pool for addition is a duplicate of an existing pool so that the pool addition is only successful when there is no duplicate. This can be done by using a mapping of `addresses` -> `booleans`, which can restrict the same address from being added twice. In addition, consider not using contract `MasterChef` and to use contract `MasterChefV2` instead, since `MasterChefV2` has already solved this issue by adding `nonDuplicated` modifier.

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

TDC-02 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Minor	projects/shibaswapv1/contracts/TopDog.sol: 169, 316, 322, 327, 337, 342, 347, 352, 357, 362, 154, 332, 368, 373, 378	✓ Resolved

Description

The owner of the contract with the `owner` role has the privilege to execute the following functions to update the sensitive settings of the project. Any compromise to the `owner` account may allow the hacker to manipulate the project through these functions.

- `updateRewardPerBlock()`
- `setMigrator()`
- `setRewardMintPercent()`
- `setDevRewardMintPercent()`
- `setLockingPeriod()`
- `devUpdate()`
- `tBoneBoneDistributorUpdate()`
- `xShibBoneDistributorUpdate()`
- `xLeashBoneDistributorUpdate()`
- `devPercentUpdate()`
- `tBonePercentUpdate()`
- `xShibPercentUpdate()`
- `xLeashPercentUpdate()`

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Shiba**]: The team acknowledges the issue and applied the MultiSig solution in any sensitive privilege access(i.e., `owner` role).

The Emergency Multisig members have trusted members of the Community and the Defi environment. There must be 6 out of 9 signatures from the below addresses for a transaction to be approved.

- MULTISIG EMERGENCY ADDRESS: 0x4267A3aD7d20c2396ebb0Fe72119984F7073761C
    - @OMEGA_HYPERION: 0x399EC033EE08241512212a4C388a76C9d3aB1c00
    - @KAAL_DHAIRYA: 0xBab4F3e701F6d2e009Af3C7f1eF2e7dD68225E96
    - @HYROSHI_KIPA: 0x80e32DEfc16ce8f78d09E6ef7065AfE031bAcab7
    - @JUNE_HORLA: 0x6948cBbEa74549062050a164d8fc4cFF27E82084
    - @SISLEY_ARGONAUT: 0xe166c948b8aED157575B6792019cdeE8a5177dcE
    - @COUNTER_NOMAD: 0x8E1B6Af660C14f5CC28727f23fCcBC977bd89B6B
    - @SHINATO_SAMA: 0x6b162Bc637bAAe0DAC38c200D9727fc679a0cCE4
    - @MISS_PHOENIX_SHIB: 0x30f45F7b08164D2Dd38D9Cdd8509b1E580432d04
    - @BURF_DURF: 0x5D471E3a033EaF7eE0cA303405978Da4c2cdAD33

You can find more details about the MultiSig Model include the settings and members' information in the the Shiba Inu Ecosystem Woof Paper Page 24.

TDC-03 | Over Minted Token

Category	Severity	Location	Status
Logical Issue	● Minor	projects/shibaswapv1/contracts/TopDog.sol: 235~243	✓ Resolved

Description

`updatePool()` function minted 100%(boneReward) + 10%(devBoneReward)+ 1%(tBONE)+ 3%(xSHIB)+ 1%(xLEASH) of total rewards.

Recommendation

We advise the client to fix to mint 100% of the block reward instead of 100% + 10% + 1% + 3% + 1% = 115% of the the block reward .

Alleviation

[**Shiba**]: In the latest whitepaper, rewards for tBONE, xSHIBA and xLEASH are "additionally" minted, which means the percentages are calculated based on the amount of `boneReward` rather than that of all rewards, so the aforementioned percentages are correct.

TDC-04 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Logical Issue	● Minor	projects/shibaswapv1/contracts/TopDog.sol: 250, 273	✓ Resolved

Description

When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. As a result, an inconsistency in the amount will occur and the transaction may fail due to the validation checks.

Recommendation

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

Alleviation

[Shiba]: The team reviewed the issue and disagreed with the description. The team confirmed that the `TopDog` contract would not support any external deflationary tokens.

[CertiK]: We agreed that the issue wouldn't occur if the token contract does not support any deflationary tokens.

TDC-05 | Lack of Event Emission for Significant Transactions

Category	Severity	Location	Status
Coding Style	● Informational	projects/shibaswapv1/contracts/TopDog.sol: 154, 169, 316, 322, 327, 332 , 337, 342, 347, 352, 357, 362, 368, 373, 378	✓ Resolved

Description

The function that affects the status of sensitive variables should be able to emit events as notifications to customers.

- `updateRewardPerBlock()`
- `setMigrator()`
- `setRewardMintPercent()`
- `setDevRewardMintPercent()`
- `setLockingPeriod()`
- `devUpdate()`
- `tBoneBoneDistributorUpdate()`
- `xShibBoneDistributorUpdate()`
- `xLeashBoneDistributorUpdate()`
- `devPercentUpdate()`
- `tBonePercentUpdate()`
- `xShibPercentUpdate()`
- `xLeashPercentUpdate()`

Recommendation

We advise the client to consider adding events for the above-mentioned sensitive actions and emit them in the function.

```
event SetDev(address indexed user, address indexed _devaddr);

function devUpdate(address _devaddr) public onlyOwner {
    devaddr = _devaddr;
     emit SetDev(msg.sender, _devaddr);
}
```

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

TDC-06 | Misleading Result of Multiplier Calculation

Category	Severity	Location	Status
Logical Issue	● Minor	projects/shibaswapv1/contracts/TopDog.sol: 186	✓ Resolved

Description

In the function `TopDog.getMultiplier()`, the multiplier should be calculated by the following formula:

```
    (number of blocks with bonus) * BONUS_MULTIPLIER + (number of blocks without bonus)
```

However, `startBlock`, which is set in L121, is not considered in the calculation.

When the function is triggered by `TopDog.pendingBone()` (L205) and `TopDog.updatePool()` (L231), the input variables `_from` and `_to` are always greater than `startBlock`, so it is unnecessary to consider `startBlock`.

Given the `TopDog.getMultiplier()` is a public function, which means it can be called externally, all possibilities of the input need to be fully considered.

For example, if `_from < startBlock` and `_to < startBlock`, the multiplier should be `_from.sub(startBlock).mul(BONUS_MULTIPLIER)`, rather than `_to.sub(_from).mul(BONUS_MULTIPLIER)` which is calculated in the function.

Recommendation

We advise the client to use `startBlock` in the calculation for the multiplier if `_from < startBlock`:

```
    function getMultiplier(uint256 _from, uint256 _to) public view returns (uint256) {
        if (_from < startBlock) {
            _from = startBlock;
        }
        if (_to <= bonusEndBlock) {
            return _to.sub(_from).mul(BONUS_MULTIPLIER);
        } else if (_from >= bonusEndBlock) {
            return _to.sub(_from);
        } else {
            return bonusEndBlock.sub(_from).mul(BONUS_MULTIPLIER).add(
                _to.sub(bonusEndBlock)
            );
        }
    }
```

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

TDC-07 | Inconsistent Checks-effects-interactions Pattern

Category	Severity	Location	Status
Logical Issue	● Major	projects/shibaswapv1/contracts/TopDog.sol: 250, 273	✓ 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.

The function `deposit()` and `withdraw()` in the `TopDog` contract has state `user.rewardDebt` updated after the external call `pool.lpToken.safeTransferFrom()` and thus are vulnerable to reentrancy attacks.

For example, a user calls `TopDog.deposit()` and claims his reward calculated in L255. If `pool.lpToken.safeTransferFrom()` (L265) allows external calls defined by users, the user can re-enter `TopDog.deposit()` before `user.rewardDebt` is updated (L268). Then the user is able to claim reward again (L255) because `user.rewardDebt` is not updated.

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

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

TDC-08 | Potential Loss of Pool Rewards

Category	Severity	Location	Status
Logical Issue	● Minor	projects/shibaswapv1/contracts/TopDog.sol: 139, 160	✓ Resolved

Description

```
function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
        ...
}
```

In the function `TopDog.add()` and `TopDog.set()`, the flag '_withUpdate' determines if all the pools will be updated. This reliance might lead to significant loss of the reward.

For instance, assume we had only one pool with `pool.allocPoint == 50` and `totalAllocPoint == 50` at the beginning. Now we want to add another pool with `pool.allocPoint == 50`. There will be two scenarios on calculating the pool reward,

Case 1: _withUpdate is `true` value.

- Step 1, distribute the reward and update the pool.
- Step 2, add or set the given pool information.

(Notes: This is important because the functions update `totalAllocPoint`, which is used in calculation of pool rewards in the function `TopDog.updatePool()` (L232))

Case 2: _withUpdate is `false` value.

- Step 1, add or set the given pool information.

(Note: the pools update will happens later)

- If we call `TopDog.add()` with `_withUpdate == true`, reward for the first pool is updated and `boneReward` in L232 is `multiplier.mul(bonePerBlock)`.
- If we call `TopDog.add()` with `_withUpdate == false`, reward for the first pool is not updated before the second pool is added. Then we call `TopDog.updatePool()` to update the reward for the first pool. `boneReward` in L232 becomes `multiplier.mul(bonePerBlock).mul(50).div(100)` because

the second pool is sharing rewards with the first one. The amount of reward becomes half as much as that in the first case.

Recommendation

We advise the client to remove the `_withUpdate` flag and always update pool rewards before updating pool information.

Alleviation

[Shiba]: The team reviewed the issue and disagreed on it. The team confirmed that the flag `_withUpdate` design intended to work well with the Shiba team business flow. The `_withUpdate` flag will operate appropriately:

- The `add()` function is only `onlyOwner` access; the operation will conduct only by authorized people.
- The flag `_withUpdate` can save gas when adding multiple pools successively, such as during the launch. The team believes it's beneficial to have the option of not updating the pools each time. For instance, when adding, say, 10 pools successively, with the first 9 of them as false, and the last one as true, will enable them to start accruing rewards simultaneously, which won't be possible if we force update pools each time.

TFC-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Minor	projects/shibaswapv1/contracts/TreasureFinder.sol: 327	✓ Resolved

Description

The owner of the contract with the `owner` role has the privilege to update the address of `topCoinDestination`, which will affect the destination where the assets would be sent to. Any compromise to the account `owner` may allow the hacker to take advantage of it and transfer all withdrawn tokens to an arbitrary address/pair address.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Shiba**]: The team acknowledges the issue and applied the MultiSig solution in any sensitive privilege access(i.e., `owner` role).

The Emergency Multisig members have trusted members of the Community and the Defi environment. There must be 6 out of 9 signatures from the below addresses for a transaction to be approved.

- MULTISIG EMERGENCY ADDRESS: 0x4267A3aD7d20c2396ebb0Fe72119984F7073761C
  - @OMEGA_HYPERION: 0x399EC033EE08241512212a4C388a76C9d3aB1c00
  - @KAAL_DHAIRYA: 0xBab4F3e701F6d2e009Af3C7f1eF2e7dD68225E96
  - @HYROSHI_KIPA: 0x80e32DEfc16ce8f78d09E6ef7065AfE031bAcab7
  - @JUNE_HORLA: 0x6948cBbEa74549062050a164d8fc4cFF27E82084
  - @SISLEY_ARGONAUT: 0xe166c948b8aED157575B6792019cdeE8a5177dcE

  - @COUNTER_NOMAD: 0x8E1B6Af660C14f5CC28727f23fCcBC977bd89B6B
  - @SHINATO_SAMA: 0x6b162Bc637bAAe0DAC38c200D9727fc679a0cCE4
  - @MISS_PHOENIX_SHIB: 0x30f45F7b08164D2Dd38D9Cdd8509b1E580432d04
  - @BURF_DURF: 0x5D471E3a033EaF7eE0cA303405978Da4c2cdAD33

You can find more details about the MultiSig Model include the settings and members' information in the the Shiba Inu Ecosystem Woof Paper Page 24.

TFC-02 | Lack of Event Emission for Significant Transactions

Category	Severity	Location	Status
Coding Style	● Informational	projects/shibaswapv1/contracts/TreasureFinder.sol: 327	✓ Resolved

Description

The function that affects the status of sensitive variables should be able to emit events as notifications to customers

Recommendation

Consider adding events for sensitive actions, and emit them in the function like below:

```
event SetTopCinDestination(address indexed user, address indexed _adminAddress);

function setTopCoinDestination(address _destination) external onlyOwner {
    topCoinDestination = _destination;
    emit SetTopCinDestination(msg.sender, _destination)
}
```

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

TFC-03 | Potential Sandwich Attack

Category	Severity	Location	Status
Logical Issue	● Minor	projects/shibaswapv1/contracts/TreasureFinder.sol: 310, 315	Partially Resolved

Description

When `pair.swap()` is triggered for a trade of asset `fromToken` for `toToken`, an attacker observing this transaction can manipulate the exchange rate by frontrunning a transaction to purchase one of the assets and make profits by backrunning a transaction to sell the asset.

Here is a possible exploit scenario: A user plans to make a transaction of swapping 100 `fromToken` for 1 `toToken`. The attacker can monitor the mempool and know the transaction detail (i.e, gas) for taking the benefit of frontrunning the victim's transaction. An attacker could raise the price of `toToken` by swapping `fromToken` for `toToken` before the transaction. As a result, the user might get less `toToken` than he expected. After the transaction, the attacker would be able to swap `toToken` for more `fromToken` than he used in his previous transaction.

Recommendation

We recommend setting a proper maximum slippage when swapping one pair of the assets.

Alleviation

[**Shiba**]: The team acknowledged this issue and decided to take the following alleviations:

- Set the bridge for low liquidity tokens to be their most liquid pair.
- Swap only highly liquid pairs, where price manipulation (without a flashloan) is not feasible.

UVF-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Major	projects/shibaswapv1/contracts/uniswapv2/UniswapV2Factory.sol: 86, 91, 96, 101, 109, 117	✓ Resolved

Description

The owner of the contract with the `owner` role has the privilege to control the following sensitive variables and functions beyond the scope of the original version of UniswapV2Factory.sol.

- `migrator` in function `setMigrator()`
- `feeToSetter` in function `setFeeToSetter()`.
- `topCoins` in function `setTopCoin()`.
- `totalFeeTopCoin`,`alphaTopCoin`, and `betaTopCoin` in function `setTopCoinFee()`.
- `totalFeeRegular`,`alphaRegular`, and `betaRegular` in function `setRegularCoinFee()`.
- Function `updatePairFee()`.

Any compromise to the account `owner` may allow the hacker to take advantage of these functions and variables, and eventually manipulate the entire project's economical system.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Shiba**]: The team acknowledges the issue and applied the MultiSig solution in any sensitive privilege access(i.e., `owner` role).

The Emergency Multisig members have trusted members of the Community and the Defi environment. There must be 6 out of 9 signatures from the below addresses for a transaction to be approved.

- MULTISIG EMERGENCY ADDRESS: 0x4267A3aD7d20c2396ebb0Fe72119984F7073761C
    - @OMEGA_HYPERION: 0x399EC033EE08241512212a4C388a76C9d3aB1c00
    - @KAAL_DHAIRYA: 0xBab4F3e701F6d2e009Af3C7f1eF2e7dD68225E96
    - @HYROSHI_KIPA: 0x80e32DEfc16ce8f78d09E6ef7065AfE031bAcab7
    - @JUNE_HORLA: 0x6948cBbEa74549062050a164d8fc4cFF27E82084
    - @SISLEY_ARGONAUT: 0xe166c948b8aED157575B6792019cdeE8a5177dcE
    - @COUNTER_NOMAD: 0x8E1B6Af660C14f5CC28727f23fCcBC977bd89B6B
    - @SHINATO_SAMA: 0x6b162Bc637bAAe0DAC38c200D9727fc679a0cCE4
    - @MISS_PHOENIX_SHIB: 0x30f45F7b08164D2Dd38D9Cdd8509b1E580432d04
    - @BURF_DURF: 0x5D471E3a033EaF7eE0cA303405978Da4c2cdAD33

You can find more details about the MultiSig Model include the settings and members' information in the the Shiba Inu Ecosystem Woof Paper Page 24.

UVF-02 | Reusable Code

Category	Severity	Location	Status
Gas Optimization	● Informational	projects/shibaswapv1/contracts/uniswapv2/UniswapV2Factory.sol: 82, 87, 92, 97, 102, 110, 118	✓ Resolved

Description

The require check `require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');` is frequently used in multiple functions.

Recommendation

The frequently used code can be converted into a modifier and be adopted in all these functions:

```
modifier checkFeeToSetter(){
    require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
    _;
}
```

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

UVF-03 | Lack of Event Emission for Significant Transactions

Category	Severity	Location	Status
Coding Style	● Informational	projects/shibaswapv1/contracts/uniswapv2/UniswapV2Factory.sol: 81, 86, 91, 96, 101, 109	✓ Resolved

Description

Function that affect the status of sensitive variables should be able to emit events as notifications to customers

Recommendation

Consider adding events for sensitive actions, and emit them in the function like below:

```
event SetFeeTo(address indexed user, address indexed _adminAddress);

function setFeeTo(address _feeTo) external override {
    require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
    feeTo = _feeTo;
     emit SetFeeTo(msg.sender, _feeTo);
}
```

Alleviation

[Shiba]: The team heeded our advice and resolved this issue in the commit `b4e8234087b1bc52f14c0e5e94115ca3fc8e47bb`.

UVP-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Major	projects/shibaswapv1/contracts/uniswapv2/UniswapV2Pair.sol: 82~84, 89~91	✓ Resolved

Description

Currently, the values of `alpha`, `beta`, and `totalFee` are not validated in the constructor of the contract. All of them should be positive and `beta` should be greater than `alpha`.

Moreover, the modification of `alpha` and `beta` may potentially affect the amount of liquidity that could be transferred to address `feeTo` and thus break the entire project economy system. The liquidity that could be transferred to address `feeTo` in function `_mintFee()` can be represented by the following equation:

$$liquidity = \frac{1}{((\beta/\alpha) * \sqrt{k_2})/(\sqrt{k_2} - \sqrt{k_1}) - 1}$$

where:

- `k_1` is the value of k before adding liquidity
- `k_2` is the value of k after adding liquidity
- α is the parameter `alpha`
- β is the parameter `beta`

Based on above equation, we can see that if and only if the value of $((\beta/\alpha) * \sqrt{k_2})/(\sqrt{k_2} - \sqrt{k_1})$ tends to 1 on the positive direction (i.e $\beta/\alpha \to ((\sqrt{k_2} - \sqrt{k_1})/\sqrt{k_2})^{+}$), the value of $liquidity$ will be an extreme large number.

Recommendation

We advise the client to add the following input validators:

```
require(_alpha > 0, "_alpha must be greater than 0");
require(_beta > _alpha, "beta should always be later than alpha");
require(_totalFee > 0, "totalFee should not be 0, which will allow free flash swap");
```

Also, we advise the client to consider the possibility of the aforementioned case before setting new values for `alpha` and `beta`.

Alleviation

[**Shiba**]: The team heeded our advice and resolved this issue in the commit `6c6fed3662f811cfe95d3b49be730ce53c65fe95`.

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.

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

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

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

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