9inch Audit

July 2023

By CoinFabrik

v202306

Executive Summary

CoinFabrik was asked to audit the contracts for the 9Inch project.

During this audit we found two critical issues, four medium issues and several minor issues.

Scope

The audited files are from the git repository located at https://github.com/9inchswap/9InchSM. The audit is based on the commit `12bfd42b293417dd32eae6f0d3e1ee93d6125c55`.

The scope for this audit includes and is limited to the following files and its dependencies in the audited commit. The audit does not include dependencies outside the repository:

- `contracts/farm/MasterChefV2.sol`
- `contracts/pool/SmartChefFactory.sol`
- `contracts/pool/TokenFlexiblePool.sol`
- `contracts/pool/TokenPool.sol`
- `contracts/swap/NineInchFactory.sol`
- `contracts/swap/NineInchPair.sol`
- `contracts/swap/NineInchRouter.sol`
- `contracts/tokens/BBC.sol`
- `contracts/tokens/NineInch.sol`

See the Audited Files Origin section for descriptions of the audited files.

No other files in this repository were audited. The rest of its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Contracts Origins

Most of the contracts included in this audit are based upon contracts in the UniswapV2 and PancakeSwap projects. As part of this audit, and based on the information provided by the development team, we tracked down the origin we believe those files have. We also documented in this section how each file differs from the upstream version.

Upstream Sources

These are the sources where the upstream code was obtained from.

Git commits:

- `v2-core`:

    - Repository: https://github.com/Uniswap/v2-core.git
    - Commit: `ee547b17853e71ed4e0101ccfd52e70d5acded58` (master top)
- `v2-periphery`:
    - Repository https://github.com/Uniswap/v2-periphery.git
    - Commit: `0335e8f7e1bd1e8d8329fd300aea2ef2f36dd19f` (master top)
- `solidity-lib`:
    - Repository: https://github.com/Uniswap/solidity-lib.git
    - Commit: `c01640b0f0f1d8a85cba8de378cc48469fcfd9a6` (master top)
- `governance`:
    - Repository: https://github.com/Uniswap/governance.git
    - Commit: `eabd8c71ad01f61fb54ed6945162021ee419998e` (master top)
- `pancakeswap-contracts`:
    - Repository: https://github.com/pancakeswap/pancake-smart-contracts.git
    - Commit: `d8f55093a43a7e8913f7730cfff3589a46f5c014` (master top)

Deployed contracts:

- `masterchef`: https://bscscan.com/address/0xa5f8C5Dbd5F286960b9d90548680aE5ebFf07652#code
- `cakepool`: https://bscscan.com/address/0x45c54210128a065de780c4b0df3d16664f7f859e#code

Audited Files Origin

Uniswap-Based Contracts

contracts/swap/NineInchERC20.sol

- **Upstream source:** `v2-core`
- **Upstream file:** `contracts/UniswapV2ERC20.sol`

Functional changes:

- Some names were changed from UniswapV2 to 9Inch in the token name, symbol and error messages.
- The safe-math library import path was changed.

contracts/swap/NineInchFactory.sol

- **Upstream source:** `v2-core`
- **Upstream file:** `contracts/UniswapV2Factory.sol`

Functional changes:

- Some names were changed from UniswapV2 to 9Inch in error messages, imports, and types.

- The `INIT_CODE_PAIR_HASH` constant was added, but it is not being used in any solidity code in the project.

contracts/swap/NineInchPair.sol

- **Upstream source:** `v2-core`
- **Upstream file:** `contracts/UniswapV2Pair.sol`

Functional changes:

- Some import statements were changed, but we could not find any functional change associated with it.
- Some names were changed from UniswapV2 to 9Inch in imports, types and error messages.
- The `_mintFee()` and `swap()` functions were changed by changing some magic[1] constants in the code.

contracts/swap/NineInchRouter.sol

- **Upstream source:** `v2-periphery`
- **Upstream file:** `contracts/UniswapV2Router02.sol`

Functional changes:

- Some import statements were changed, but we could not find any functional change associated with it.
- Some names were changed from UniswapV2 to 9Inch in imports, types and error messages.

contracts/tokens/NineInch.sol

- **Upstream source:** `governance`
- **Upstream file:** `contracts/Uni.sol`

The file has too many changes, it has been analyzed as if it was written from scratch.

The `NineInch` contract provides an enhanced ERC20 token with voting delegation capabilities, ensuring transparency and flexibility in voting mechanisms.

Uniswap-based Dependencies

contracts/swap/interfaces/INineInchCallee.sol

- **Upstream source:** `v2-core`
- **Upstream file:** `contracts/interfaces/IUniswapV2Callee.sol`

Functional changes:

- The interface name was changed to be part of the 9Inch project.

[1] See https://en.wikipedia.org/wiki/Magic_number_(programming)

contracts/swap/interfaces/INineInchERC20.sol

- **Upstream source:** `v2-core`
- **Upstream file:** `contracts/interfaces/IUniswapV2ERC20.sol`

Functional changes:

- The interface name was changed to be part of the 9Inch project.

contracts/swap/interfaces/INineInchFactory.sol

- **Upstream source:** `v2-core`
- **Upstream file:** `contracts/interfaces/IUniswapV2Factory.sol`

Functional changes:

- The interface name was changed to be part of the 9Inch project.

contracts/swap/interfaces/INineInchPair.sol

- **Upstream source:** `v2-core`
- **Upstream file:** `contracts/interfaces/IUniswapV2Pair.sol`

Functional changes:

- The interface name was changed to be part of the 9Inch project.

contracts/swap/interfaces/INineInchRouter01.sol

- **Upstream source:** `v2-periphery`
- **Upstream file:** `contracts/interfaces/IUniswapV2Router01.sol`

Functional changes:

- The interface name was changed to be part of the 9Inch project.

contracts/swap/interfaces/INineInchRouter02.sol

- **Upstream source:** `v2-periphery`
- **Upstream file:** `contracts/interfaces/IUniswapV2Router02.sol`

Functional changes:

- The interface name was changed to be part of the 9Inch project.

contracts/swap/interfaces/IWETH.sol

- **Upstream source:** `v2-periphery`
- **Upstream file:** `contracts/interfaces/IWETH.sol`

This file has no functional changes.

contracts/libraries/SafeMath.sol

- **Upstream source:** `masterchef`

- **Upstream file:** `SafeMath.sol`

Functional changes:
- Two functions, `min()` and `sqrt()` were added.
- The solidity pragma was changed from `^0.6.0` to `>=0.4.0`.

It must also be noted that this code is functionally a super set of the `contracts/libraries/SafeMath.sol` file in the `v2-periphery` source. All the functions defined there are implemented in this file, and the only functional changes in those functions are different error messages.

contracts/libraries/Math.sol

- **Upstream source:** `v2-core`
- **Upstream file:** `contracts/libraries/Math.sol`

This file has no functional changes.

contracts/libraries/UQ112x112.sol

- **Upstream source:** `v2-core`
- **Upstream file:** `contracts/libraries/UQ112x112.sol`

This file has no functional changes.

contracts/libraries/NineInchLibrary.sol

- **Upstream source:** `v2-periphery`
- **Upstream file:** `contracts/libraries/UniswapV2Library.sol`

Functional changes:
- A magic constant was changed in the `pairFor()` function.
- The solidity pragma was changed from `>=0.5.0` to `0.6.6`.
- Some names were changed from UniswapV2 to 9Inch in imports, types and error messages.

contracts/libraries/TransferHelper.sol

- **Upstream source:** `solidity-lib`
- **Upstream file:** `contracts/libraries/TransferHelper.sol`

Functional changes:
- The solidity pragma was changed from `>=0.6.0` to `>=0.6.6`.
- Some error messages were changed.
- `uint256` was changed to `uint`.

PancakeSwap-Based Contracts

contracts/farm/MasterChefV2.sol

- **Upstream source:** `masterchef`
- **Upstream file:** `MasterChefV2.sol`

Functional changes:

- The solidity pragma was changed from `0.6.12` to `>=0.6.12`;
- The ReentrancyGuard contract was copied in the file, instead of importing it from `@openzeppelin/contracts/utils/ReentrancyGuard.sol`.
- The cake contract was renamed as BBC. This implies changes in comments, type names, field names, variable names and constant names.
- Removed legacy functionality associated with `MasterchefV1`.
- Added the `transferOwnershipofBBC()` function, only invocable by the owner of the contract.

contracts/pool/TokenPool.sol

- **Upstream source:** `cakepool`
- **Upstream file:** `CakePool.sol`

The file has too many changes, it has been analyzed as if it was written from scratch.

`TokenPool` is a token pool management contract providing detailed control over parameters, fee calculations, and user-based operations like deposits and withdrawals.

contracts/pool/TokenFlexiblePool.sol

- **Upstream source:** `cakepool`
- **Upstream file:** `CakePool.sol`

The file has too many changes, it has been analyzed as if it was written from scratch.

`TokenFlexiblePool` is a façade over another token pool, exposing a different API. simplified version of the `TokenPool` contract, delivering essentially the same user operations, without the granularity over parameters and a feature to boost withdrawal amounts.

contracts/pool/SmartChefFactory.sol

- **Upstream source:** `pancakeswap-contracts`
- **Upstream file:** `projects/smartchef/v2/contracts/SmartChefInitializable.sol`

Functional changes:

- The abicoder pragma was removed.
- Import paths were changed.

- The `"./interfaces/IPancakeProfile.sol"` import was removed.
- Some variables were renamed, including changes to reflect that the token is named BBC instead of CAKE.
- All constructor parameters were removed.
- Some error messages were changed.
- Pancake profile functionality was removed.

It must be noted that the `projects/smartchef/v2/contracts/SmartChefFactory.sol` exists in the upstream source, but the contents of this file correspond to `projects/smartchef/v2/contracts/SmartChefInitializable.sol`.

PancakeSwap-Based Dependencies

contracts/libraries/Address.sol

- **Upstream source:** `cakepool`
- **Upstream file:** `Address.sol`

Both the upstream file and the project file appear to be old and different versions of the openzeppelin's address library. The only used function is `functionCall()`, invoked in the `contracts/tokens/SafeBEP20.sol` file.

contracts/tokens/BEP20.sol

- **Upstream source:** `pancakeswap-contracts`
- **Upstream file:** `projects/bsc-library/contracts/BEP20.sol`

Functional changes:

- The `mint()` function has been declared `virtual`.
- Some import paths were changed.

contracts/tokens/SafeBEP20.sol

- **Upstream source:** `pancakeswap-contracts`
- **Upstream file:** `projects/bsc-library/contracts/SafeBEP20.sol`

Functional changes:

- Some import paths were changed.

contracts/tokens/interfaces/IBEP20.sol

- **Upstream source:** `pancakeswap-contracts`
- **Upstream file:** `projects/bsc-library/contracts/SafeBEP20.sol`

Functional changes:

- The `mint()` function was added.

contracts/farm/interfaces/IMasterChefV2.sol

- **Upstream source:** `cakepool`
- **Upstream file:** `IMasterChefV2.sol`

Functional changes:

- The `pendingCake()` function was renamed as `pendingBBC()`.
- The `userInfo()` function returns 3 `uint256` values (it returns 2 uint256 values in the upstream source).
- The `BBC()`, `lpToken()`, `poolLength()`, `getBoostMultiplier()` and `updateBoostMultiplier()` functions were added.

No-Upstream Contracts

contracts/tokens/BBC.sol

ERC20 contract made with the BEP20 library, which is taken from the `pancakeswap-contracts` upstream source.

No-Upstream Dependencies

contracts/pool/interfaces/ITokenPool.sol

This interface definition is imported in `contracts/pool/TokenFlexiblePool.sol`.

Methodology

CoinFabrik was provided with the source code and documentation about the upstream code used. Six auditor-weeks were spend auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses.

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling

- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

It must be noted that on contracts where only small changes were made, we assumed that the upstream contract is correct. See Contracts Origins.

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

ID	Title	Severity	Status
CR-01	Infinite Vote Creation	Critical	Unresolved
CR-02	Price Decrease DOS	Critical	Unresolved
ME-01	Needs Rewrite	Medium	Unresolved
ME-02	Possible Reentrancy Attacks in TokenFlexiblePool	Medium	Unresolved
ME-03	Solidity Pragmas	Medium	Unresolved
ME-04	Possible Reentrancy Attacks in TokenPool	Medium	Unresolved
MI-01	No Zero Checks	Minor	Unresolved
MI-02	Excessive Allowance	Minor	Unresolved
MI-03	No Front-Running Protection	Minor	Unresolved
MI-04	Fee-Avoidance	Minor	Unresolved

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

CR-01 Infinite Vote Creation

Location:

- `contracts/tokens/NineInch.sol`

Votes, as obtained by the `NineInch.getCurrentVotes()` function, can be grown infinitely. Any token holder can obtain an unbounded number of votes.

Hereunder is a proof of concept showing the issue:

```
it("Create votes cycling tokens (Σvotes > Σtokens)", async function () {
await nineInch
    .connect(owner)
    .mint(addr0.address, ethers.utils.parseEther("100"));

await nineInch.connect(addr0).delegate(addr3.address);

await nineInch
    .connect(addr0)
    .transfer(addr1.address, ethers.utils.parseEther("100"));

await nineInch.connect(addr1).delegate(addr2.address);

await nineInch
    .connect(addr1)
    .transfer(addr0.address, ethers.utils.parseEther("100"));

sumOfDelegatedVotes = (await nineInch.getCurrentVotes(addr0.address))
    .add(await nineInch.getCurrentVotes(addr1.address))
    .add(await nineInch.getCurrentVotes(addr2.address))
    .add(await nineInch.getCurrentVotes(addr3.address));

expect(sumOfDelegatedVotes).to.be.above(
    await nineInch.balanceOf(addr0.address)
);
});
```

Please note that this bug is not present in the upstream code.

It must also be noted that, while not strictly this issue, it is also a problem that burning some tokens does not destroy the votes associated with them.

Recommendation

Either remove the functionality to delegate votes altogether or make sure that for each delegation the total number of votes stays the same, and that total number of votes are only increased when minting, the total number of votes stays the same on transfers and that the total number of votes decreases when burning. Also a mechanism to undo a delegation could be a good addition.

Status

Unresolved.

CR-02 Price Decrease DOS

Location:

- `contracts/pool/TokenFlexiblePool.sol: 118-119,202-203`

When doing a deposit or a withdrawal in the `TokenFlexiblePool` contract, if the share price decreases since the last user action, the operation will revert with an underflow error. The decrease can happen if `isPureBBCPool()` is `true`. The code is repeated in both operations:

```
uint256 totalAmount = (user.shares * balanceOf()) / totalShares;
uint256 earnAmount = totalAmount - user.bbcAtLastUserAction;
```

The underflow is triggered as `totalAmount` will be less than `user.bbcAtLastUserAction`. This, in turn, impedes the user from either depositing or withdrawing until the price reaches the original level again.

Recommendation

This issue should be fixed by a redesign of the `TokenFlexiblePool` contract. Designing this new logic is out of the scope of this report because it probably requires making functional decisions. However, when completed, it should take into account the following recommendation.

`TokenFlexiblePool` contract's behavior is very different when `isPureBBCPool()` is `true`. Therefore, this contract should be split into two contracts, maybe using an abstract and common base contract. When writing the new contract for the case where `isPureBBCPool() == true`, the logic should handle eventual price decreases (which is not needed when `isPureBBCPool() == false`). This new logic might imply not paying performance fees until the funds are withdrawn.

Status

Unresolved.

Medium Severity Issues

ME-01 Needs Rewrite

Copying and pasting old code from the internet is not a good security practice nor a good software engineering practice. Even more so when the code is altered extensively afterwards. Most of the code analyzed in this audit suffers from this. This is compounded by:

1. A part of the copied code should not be used, as explained in its documentation. See `projects/bsc-library/README.md` in the `pancakeswap-contracts` upstream source.
2. Some solidity pragmas were changed. There are several incompatible changes in solidity version changes.
3. Some dependencies were shared between the different upstream sources, but different versions were used in the upstream code. The development team in some

instances just kept one and used it for all the code. There is no warranty that this approach will work properly.
4. The version control (git) has not been used properly. Looking at the history of the development it is not possible to see the original code used and the changes applied.
5. The new code suffers from serious design problems. For example the `TokenPool` and `TokenFlexiblePool` contracts do not follow the single responsibility principle[2]. This can be seen at least 20 times where the `isPureBBCPool()` function, which by the way is copy pasted in the two contracts, is invoked to choose different logic paths.
6. There is commented code in the audited code. For example, we found commented code at `contracts/pool/TokenFlexiblePool.sol:151,221`, `contracts/tokens/BBC.sol:9,25`, `contracts/tokens/NineInch.sol:35` and `contracts/farm/MasterChefV2.sol:223-228,683-687`. This list is not intended to be exhaustive.
7. There are functions only intended for development purposes in the audited code. For example, the `MasterChefV2.transferOwnershipOfBBC()` function is present in the code but documented to be only for development in the `Audit Full Report.pdf` file. The code shall include just the logic needed for the production environment.

It also must be noted that the GPLv3 license for Uniswap is being broken by relicensing the code as MIT.

Please note that CR-01 Infinite Vote Creation and CR-02 Price Decrease DOS issues can be attributed to this issue as the root cause, but fixing them is not enough to solve the underlying software development problem noted here.

Recommendation

A full rewrite from scratch, properly following standard software engineering practices, is required. Good software engineering practices include test-first development, using libraries instead of copy and pasting code, proper use of a version control system and following the DRY and YAGNI principles. Given that the code needs to be rewritten, we recommend simplifying it. Doing a code collage naturally increases the attack surface of the system. When this rewrite is made, use a single and up to date solidity version and pin it in the source code.

Status

Unresolved.

ME-02 Possible Reentrancy Attacks in `TokenFlexiblePool`

Location:

[2] https://en.wikipedia.org/wiki/Single-responsibility_principle

- `contracts/pool/TokenFlexiblePool.sol`

If the `bbcPool`, `token` or `bbc` contracts are forced to make a call to an attacker contract while in the activation of the `deposit()`, `withdraw()` or `claim()` functions in the `TokenFlexiblePool` contract, or the `treasury` account is compromised, then a reentrancy attack can be made, potentially losing funds.

Recommendation

The `deposit()`, `withdraw()` and `claim()` functions in the `TokenFlexiblePool` contract need to either be marked as `nonReentrant` or written following the checks-effects-interaction pattern. Another option is to provide a mathematical proof that this attack is not possible.

Status

Unresolved.

ME-03 Solidity Pragmas

Several solidity pragmas are used in the audited code. Such as:

1. `pragma solidity >=0.6.6;` (in `contracts/libraries/Address.sol:3`)
2. `pragma solidity 0.6.6;` (in `contracts/libraries/NineInchLibrary.sol:2`)
3. `pragma solidity >=0.4.0;` (in `contracts/libraries/SafeMath.sol:3`)
4. `pragma solidity ^0.8.0;` (in `contracts/pool/SmartChefFactory.sol:2`)
5. `pragma solidity =0.5.16;` (in `contracts/swap/NineInchFactory.sol:2`)
6. `pragma solidity >=0.6.2;` (in contracts/swap/`interfaces/INineInchRouter02.sol:2`)
7. `pragma solidity >=0.5.0;` (in `contracts/swap/interfaces/IWETH.sol:2`)

Using any of those solidity pragmas is not recommended. Using old solidity versions may lead to bugs, including security ones. Mixing different solidity versions in the same project may lead to bugs, including security ones. Not pinning the solidity version to a single version may lead to bugs, including security ones.

Recommendation

Choose a single solidity version for the project. Pin the solidity version in the code (no > or ~ in the solidity pragma). Rewrite the code to use the features in the solidity version and do all the changes required to avoid incompatibilities between solidity versions.

Status

Unresolved.

ME-04 Possible Reentrancy Attacks in `TokenPool`

Location:

- `contracts/pool/TokenPool.sol`

If the `masterchefV2`, `token` or `bbc` contracts are forced to make a call to an attacker contract while in the activation of the `deposit()`, `withdraw()`, `withdrawByAmount()`, `unlock()` or `claim()` functions in the `TokenPool` contract then a reentrancy attack can be made, potentially losing funds.

Recommendation
The `deposit()`, `withdraw()`, `withdrawByAmount()`, `unlock()` and `claim()` functions in the `TokenPool` contract need to either be marked as `nonReentrant` or written following the checks-effects-interaction pattern. Another option is to provide a mathematical proof that this attack is not possible.

Status
Unresolved.

Minor Severity Issues

MI-01 No Zero Checks

Location:
- `contracts/swap/NineInchRouter.sol: 23-26`
- `contracts/pool/TokenFlexiblePool.sol: 68-81`
- `contracts/pool/TokenPool.sol: 107-122,129-138`
- `contracts/farm/MasterChefV2.sol: 201-204`
- `contracts/swap/NineInchFactory.sol: 25,27,58-61,63-66`
- `contracts/swap/NineInchPair.sol: 88-92,`

It is a standard best practice to check that all the addresses passed to constructors or functions are not zero. This practice was not implemented in lots of places in the code.

Recommendation
Check that all the passed addresses are non-zero via `require` statements.

Status
Unresolved.

MI-02 Excessive Allowance

Location:
- `contracts/pool/TokenFlexiblePool.sol:80`

In the `TokenFlexiblePool` constructor, the `bbcPool` is given the ability to withdraw infinite tokens. This is a bad security practice, as it may potentially allow an attacker to take control of all the tokens in the contract.

Recommendation
Use the `token.safeIncreaseAllowance()` function to award the `bbcPool` contract the required allowance only when needed, passing the actual amount to be withdrawn.

Status
Unresolved.

MI-03 No Front-Running Protection

Location:
- `contracts/pool/TokenFlexiblePool.sol`
- `contracts/pool/TokenPool.sol`

An EOA interacting with the `TokenPool` or `TokenFlexiblePool` contracts may lose an unbounded amount when doing deposit or withdrawal operations. This lack of protection may be used in a front-running attack.

Recommendation
For each withdrawal or deposit operation add an additional parameter that bounds the amount of shares or tokens (depending on the function) involved to limit the loss that can occur due to price fluctuations.

Just as an example, the `TokenPool.withdraw()` function signature can be changed to function `withdraw(uint256 _shares, uint256 _minAmount)` where `_minAmount` is the minimum amount to be withdrawn.

Another way to solve this problem is adding new functions to specify those limits (minimum or maximum depending on the case). This approach could be preferable if there is a need of preserving the current API.

Status
Unresolved.

MI-04 Fee-Avoidance

Location:
- `contracts/pool/TokenPool.sol: 218-220,467-469,509-511,758-760,824-826`

If the performance fee or the withdrawal fee for contracts is lower than the corresponding EOA fee, an attacker may pay less fees by using the `TokenPool` contract via an intermediate contract.

Recommendation
Either make sure that the EOA fees are lower than the contract fees or remove the fee distinction between EOAs and contracts altogether.

Status
Unresolved.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Centralization

BBC

The owner of the `BBC` contract can mint tokens at any time.

TokenPool

The owner of the `TokenPool` contract can set the treasury address, where all the fees are sent. They can also set the contract's admin and the contract's operator.

The admin of the `TokenPool` contract can set lots of configuration parameters that dictates fees and other settings, withdraw ERC20 tokens sent to the contract, and pause or restart the contract.

The operator of the `TokenPool` contract can unlock funds for any user.

TokenFlexiblePool

The owner of the `TokenFlexiblePool` contract can set the treasury address, where all the fees are sent. They can also set the contract's admin.

The admin of the `TokenFlexiblePool` contract can set lots of configuration parameters that dictates fees and other settings, withdraw ERC20 tokens sent to the contract and pause or restart the contract.

Upgrades

None of the audited contracts has code-upgrade previsions.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

BBC

Owner

The owner of this contract can:

- mint tokens for itself or others via the `mint()` and `mintFor()` functions.
- transfer BBC tokens awarded to the contract itself via the `transferOwnershipOfBBC()` function.
- renounce its ownership, via the `renounceOwnership()` function. If this happens, no more tokens can be minted.
- set a new owner via the `transferOwnership()` function.

By default, the owner of this contract is the deployer of the contract.

TokenFlexiblePool

Owner

The owner of this contract can:

- set the contract admin via the `setAdmin()` function.
- set the contract treasury via the `setTreasury()` function.
- renounce its ownership, via the `renounceOwnership()` function.
- set a new owner via the `transferOwnership()` function.

By default, the owner of this contract is the deployer of the contract.

Admin

The admin of this contract can:

- set the performance fee via the `setPerformanceFee()` function.
- set the withdrawal fee via the `setWithdrawFee()` function.
- set the withdrawal fee period via the `setWithdrawFeePeriod()` function.
- set the withdrawal amount booster via the `setWithdrawAmountBooster()` function.
- withdraw all the `bbcPool` funds via the `emergencyWithdraw()` function. After this function is called no more deposits are accepted and the funds to withdraw are awarded to this contract.
- withdraw any token but the staking token[3] sent to the contract via the `inCaseTokensGetStuck()` function.
- pause and restart the contract via the `pause()` and `unpause()` functions.

[3] `token` in the source code.

The initial `admin` is passed to the contract via the contract constructor in the `_admin` parameter and can be changed by the contract's owner.

TokenPool

Owner

The owner of this contract can:

- init the contract via the `init()` function
- do an emergency withdrawal in the `masterchefV2` contract via the `close()` function.
- set the contract admin via the `setAdmin()` function.
- set the contract treasury via the `setTreasury()` function.
- set the contract operator via the `setOperator()` function.
- renounce its ownership, via the `renounceOwnership()` function.
- set a new owner via the `transferOwnership()` function.

By default, the owner of this contract is the deployer of the contract.

Admin

The admin of this contract can:

- exclude or reinclude an address from charging performance fees via the `setFreePerformanceFeeUser()` function. Performance fees are charged to an address by default.
- exclude or reinclude an address from charging overdue fees via the `setOverdueFeeUser()` function. Overdue fees are charged to an address by default.
- exclude or reinclude an address from charging withdrawal fees via the `setWithdrawFeeUser()` function. Withdraw fees are charged to an address by default.
- set the performance fee for EOAs via the `setPerformanceFee()` function.
- set the performance fee for contracts via the `setPerformanceFeeContract()` function.
- set the withdrawal fee for EOAs via the `setWithdrawFee()` function.
- set the withdrawal fee for contracts via the `setWithdrawFeeContract()` function.
- set the withdrawal fee period via the `setWithdrawFeePeriod()` function.
- set the overdue fee via the `setOverdueFee()` function.
- set the max-lock duration via the `setMaxLockDuration()` function.
- set the duration factor via the `setDurationFactor()` function.
- set the duration factor when overdue via the `setDurationFactorOverdue()` function.
- set the duration of the unlock-free period via the `setUnlockFreeDuration()` function.
- set the boost weight via the `setBoostWeight()` function.

- withdraw any ERC-20 non-deposit tokens awarded to this contract via the `inCaseTokensGetStuck()` function.
- pause and restart the contract via the `pause()` and `unpause()` functions.

The initial admin is passed to the contract via the contract constructor in the `_admin` parameter and can be changed by the contract's owner.

Operator

The operator of this contract can unlock funds for any user via the `unlock()` function.

The initial operator is passed to the contract via the contract constructor in the `_operator` parameter and can be changed by the contract's owner.

Changelog

- 2023-07-14 – Initial report based on commit `12bfd42b293417dd32eae6f0d3e1ee93d6125c55`.
- 2023-07-19 – Clarify issues CR-02, ME-01 and MI-03.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the 9Inch project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.