BaryonFarm

Smart Contract Audit Report
Prepared for Baryon Network

Date Issued:	May 25, 2022
Project ID:	AUDIT2022031
Version:	v1.0
Confidentiality Level:	Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2022031
Version	v1.0
Client	Baryon Network
Project	BaryonFarm
Auditor(s)	Patipon Suwanbol Darunphop Pengkumta Fungkiat Phadejtaku
Author(s)	Wachirawit Kanpanluk
Reviewer	Natsasit Jirathammanuwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	May 25, 2022	Full report	Wachirawit Kanpanluk

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary

As requested by Baryon Network, Inspex team conducted an audit to verify the security posture of the BaryonFarm smart contracts between May 10, 2022 and May 11, 2022. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of BaryonFarm smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 1 high, 1 medium, 1 low, 1 very low, and 3 info-severity issues. With the project team's prompt response in resolving the issues found by Inspex, all issues were resolved or mitigated in the reassessment. Therefore, Inspex trusts that BaryonFarm smart contracts have high-level protections in place to be safe from most attacks.

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

Baryon Network is a MetaFi platform on BNB Chain, empowering your crypto journey with full-stack services: Swap, Earn (Farm & Stake) and Metaverse going into the future.

SmartBaryFactory is a contract that allows the platform users to stake tokens and earn rewards. The reward tokens will be stored in the SmartBaryFactoryRewarder contract and transferred to the users when they decide to claim the rewards.

Scope Information:

Project Name	BaryonFarm
Website	https://www.baryon.network/farm
Smart Contract Type	Ethereum Smart Contract
Chain	BNB
Programming Language	Solidity
Category	Yield Farming

Audit Information:

Audit Method	Whitebox
Audit Date	May 10, 2022 - May 11, 2022
Reassessment Date	May 18, 2022

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

Initial Audit: (Commit: 75f13934af5208e43922f5339833b62d015955ef)

Contract	Location (URL)
SmartBaryFactoryRewarder	https://github.com/coin98/baryon-farm/blob/75f13934af/contracts/SmartBaryFactory.sol
SmartBaryFactory	https://github.com/coin98/baryon-farm/blob/75f13934af/contracts/SmartBaryFactory.sol

Reassessment: (Commit: 426be5e620883e5a06a15fc97501323f1ee77750)

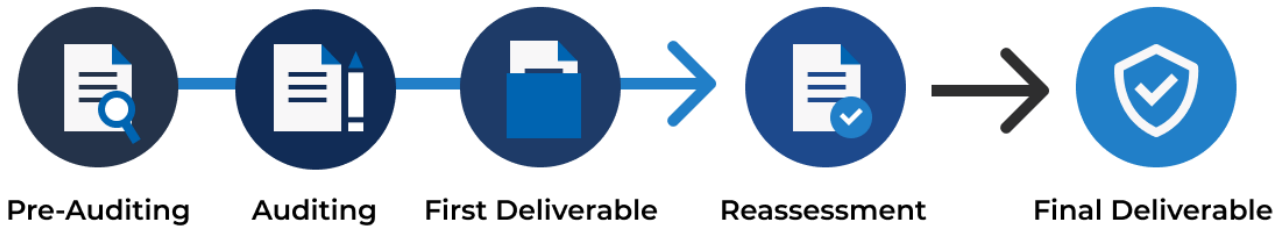
Contract	Location (URL)
SmartBaryFactoryRewarder	https://github.com/coin98/baryon-farm/blob/426be5e620/contracts/SmartBaryFactory.sol
SmartBaryFactory	https://github.com/coin98/baryon-farm/blob/426be5e620/contracts/SmartBaryFactory.sol

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The testing items checked are based on our Smart Contract Security Testing Guide (SCSTG) v1.0 (https://github.com/InspexCo/SCSTG/releases/download/v1.0/SCSTG_v1.0.pdf) which covers most prevalent risks in smart contracts. The latest version of the document can also be found at https://inspex.gitbook.io/testing-guide/.

The following audit items were checked during the auditing activity:

Testing Category	Testing Items
1. Architecture and Design	1.1. Proper measures should be used to control the modifications of smart contract logic 1.2. The latest stable compiler version should be used 1.3. The circuit breaker mechanism should not prevent users from withdrawing their funds 1.4. The smart contract source code should be publicly available 1.5. State variables should not be unfairly controlled by privileged accounts 1.6. Least privilege principle should be used for the rights of each role
2. Access Control	2.1. Contract self-destruct should not be done by unauthorized actors 2.2. Contract ownership should not be modifiable by unauthorized actors 2.3. Access control should be defined and enforced for each actor roles 2.4. Authentication measures must be able to correctly identify the user 2.5. Smart contract initialization should be done only once by an authorized party 2.6. tx.origin should not be used for authorization
3. Error Handling and Logging	3.1. Function return values should be checked to handle different results 3.2. Privileged functions or modifications of critical states should be logged 3.3. Modifier should not skip function execution without reverting
4. Business Logic	4.1. The business logic implementation should correspond to the business design 4.2. Measures should be implemented to prevent undesired effects from the ordering of transactions 4.3. msg.value should not be used in loop iteration
5. Blockchain Data	5.1. Result from random value generation should not be predictable 5.2. Spot price should not be used as a data source for price oracles 5.3. Timestamp should not be used to execute critical functions 5.4. Plain sensitive data should not be stored on-chain 5.5. Modification of array state should not be done by value 5.6. State variable should not be used without being initialized

Testing Category	Testing Items
6. External Components	6.1. Unknown external components should not be invoked 6.2. Funds should not be approved or transferred to unknown accounts 6.3. Reentrant calling should not negatively affect the contract states 6.4. Vulnerable or outdated components should not be used in the smart contract 6.5. Deprecated components that have no longer been supported should not be used in the smart contract 6.6. Delegatecall should not be used on untrusted contracts
7. Arithmetic	7.1. Values should be checked before performing arithmetic operations to prevent overflows and underflows 7.2. Explicit conversion of types should be checked to prevent unexpected results 7.3. Integer division should not be done before multiplication to prevent loss of precision
8. Denial of Services	8.1. State changing functions that loop over unbounded data structures should not be used 8.2. Unexpected revert should not make the whole smart contract unusable 8.3. Strict equalities should not cause the function to be unusable
9. Best Practices	9.1. State and function visibility should be explicitly labeled 9.2. Token implementation should comply with the standard specification 9.3. Floating pragma version should not be used 9.4. Builtin symbols should not be shadowed 9.5. Functions that are never called internally should not have public visibility 9.6. Assert statement should not be used for validating common conditions

3.3. Risk Rating

OWASP Risk Rating Methodology (https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker
- **Impact**: a measure of the damage caused by a successful attack

Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

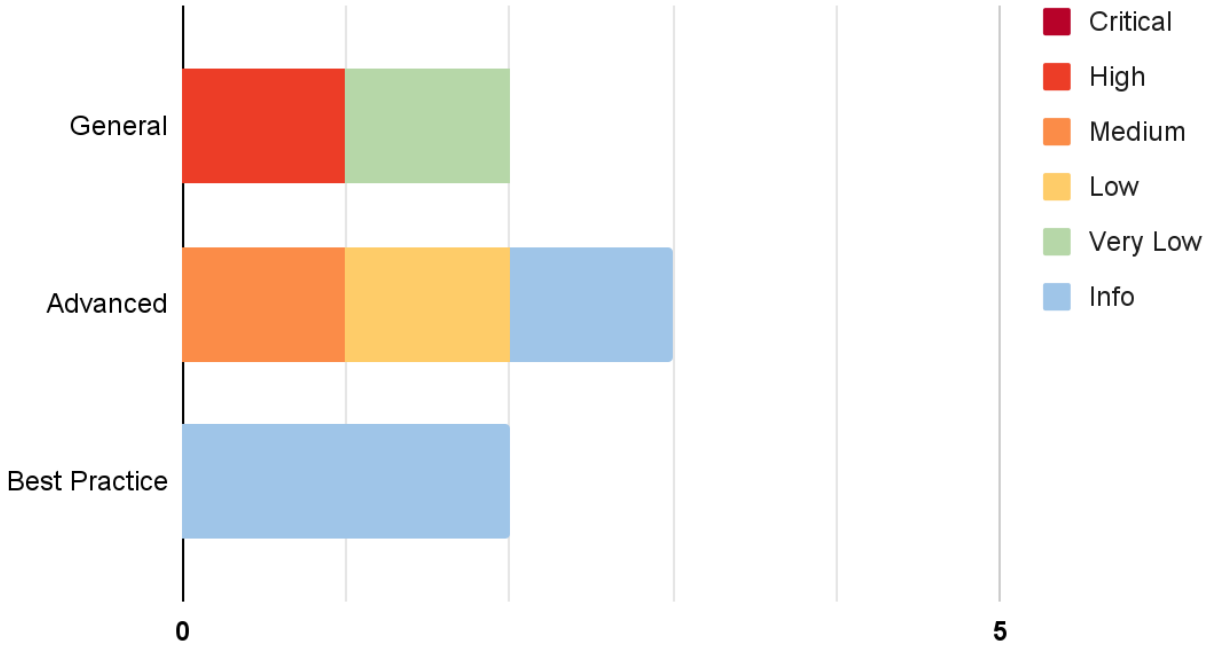
Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Impact \ Likelihood	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

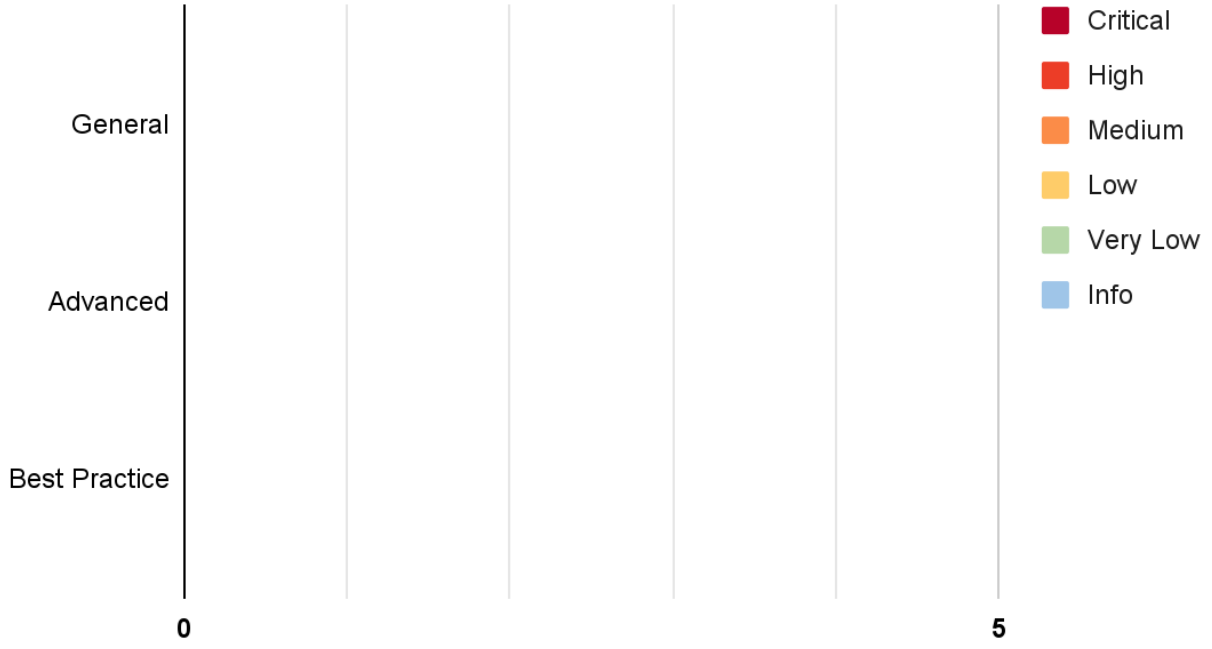
4. Summary of Findings

The following charts show the number of the issues found during the assessment and the issues acknowledged in the reassessment, categorized into three categories: **General**, **Advanced**, and **Best Practice**.

Assessment:



Reassessment:



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Title	Category	Severity	Status
IDX-001	Centralized Control of State Variable	General	High	Resolved *
IDX-002	Reward Miscalculation in updatePool() Function	Advanced	Medium	Resolved
IDX-003	Design Flaw in Reward Distribution Mechanism	Advanced	Low	Resolved
IDX-004	Insufficient Logging for Privileged Functions	General	Very Low	Resolved
IDX-005	Improper Function Visibility	Best Practice	Info	Resolved
IDX-006	Inexplicit Solidity Compiler Version	Best Practice	Info	Resolved
IDX-007	Unsupported Design for Deflationary Tokens	Advanced	Info	Resolved

* The mitigations or clarifications by Baryon Network can be found in Chapter 5.

5. Detailed Findings Information

5.1. Centralized Control of State Variable

ID	IDX-001
Target	SmartBaryFactory
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** High **Impact:** High The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users. For example, the contract owner can change the reward token or withdraw any deposit from the users immediately. **Likelihood:** Medium There is nothing to restrict the changes from being done; however, this action can only be done by the contract owner.
Status	Resolved * Baryon Network team has confirmed that the privilege function will be called through the `Timelock` contract. This means any action that would occur to the privilege function will be able to be monitored by the community conveniently. However, as the affected contracts are not yet deployed during the reassessment, the users should confirm that the contracts are under the effect of the Timelock contract before using them.

5.1.1. Description

Critical state variables can be updated at any time by the controlling authorities. Changes in these variables can cause impacts to the users, so the users should accept or be notified before these changes are effective. For example, the contract owner can execute the `withdrawMultiplePool()` function to withdraw all rewards tokens, resulting in no reward token left to be claimed by the users.

However, there is currently no constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

File	Contract	Function	Modifier
SmartBaryFactory.sol (L:561)	SmartBaryFactory	setPool()	onlyOwner
SmartBaryFactory.sol (L:792)	SmartBaryFactory	withdrawMultiplePool()	onlyOwner
SmartBaryFactory.sol (L:804)	SmartBaryFactory	withdrawPoolTokens()	onlyOwner
SmartBaryFactory.sol (L:818)	SmartBaryFactory	setRewardConfig()	onlyOwner
SmartBaryFactory.sol (L:831)	SmartBaryFactory	withdrawMultiple()	onlyOwner

5.1.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. However, if modifications are needed, Inspex suggests implementing a community-run smart contract governance to control the use of these functions.

If removing the functions or implementing the smart contract governance is not possible, Inspex suggests mitigating the risk of this issue by using a timelock mechanism to delay the changes for a reasonable amount of time, e.g., 24 hours.

However, if the timelock is used in the `SmartBaryFactory` contract, the `withdrawPoolTokens()` and the `withdrawMultiple()` functions can be excluded from the timelock by adding the following condition:

- For the `withdrawPoolTokens()` function, add a condition to prevent withdrawing a reward token. This means this function will be used to withdraw the tokens that are transferred incorrectly only.
- For the `withdrawMultiple()` function, add a condition to prevent withdrawing a staking token (`lpToken`), so the user funds will not be able to be transferred without approval by the users.

Lastly, it is suggested to change the privileged role from `onlyOwner` to another privileged role such as `onlyOperator` role to prevent the `addPool()`, `withdrawPoolTokens()`, and `withdrawMultiple()` functions from being stuck in the timelock, which can break the business design.

5.2. Reward Miscalculation in updatePool() Function

ID	IDX-002
Target	SmartBaryFactory
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Medium The rewards in the `SmartBaryFactoryRewarder` contract can be claimed largely when the pool starts or renews (active the pool again after it expires). This is because it was accumulated when the pool was added or renewed by the platform owner which should be accumulated when the pool starts to active instead. **Likelihood:** Medium It is likely that there will be a pool in which the reward distribution period does not start right away after it is added (the `pool.rewardsStartTime` is set more than 2 blocks from the `block.timestamp` when adding or setting the pool).
Status	Resolved Baryon Network team has resolved this issue as suggested by adding a condition to assign a value of `pool.lastRewardTime` as the same as the `pool.rewardStartTime` or the current block timestamp if it is greater than the `pool.rewardStartTime` in commit `426be5e620883e5a06a15fc97501323f1ee77750`.

5.2.1. Description

In the `SmartBaryFactory` contract, the platform owner can add a new pool to the contract through the `addPool()` function, allowing the users to stake the `_lpToken` and earn the rewards.

SmartBaryFactory.sol

```
472 /// @notice Add rewarder pool
473 function addPool(
474     IERC20[] memory _rewardTokens,
475     uint256[] memory _rewardMultipliers,
476     uint256 _rewardsStartTime,
477     uint256 _rewardPerSeconds,
478     uint256 _rewardsExpiration,
479     address _lpToken
480 ) external onlyOwner {
481     uint256 tokenCode;
482     assembly {
483         tokenCode := extcodesize(_lpToken)
```

```
484     }
485     IERC20 lpTokenCall = IERC20(_lpToken);
486     require(
487         tokenCode > 0 && lpTokenCall.balanceOf(address(this)) >= 0,
488         "SmartBaryFactory: Invalid token code"
489     );
490 
491     massUpdateAllPools();
492     add(
493         _rewardTokens,
494         _rewardMultipliers,
495         _rewardsStartTime,
496         _rewardPerSeconds,
497         _rewardsExpiration,
498         lpTokenCall
499     );
500 }
```

The `addPool()` function will internally call the `add()` function. It will set the `lastRewardTime` of the added pool as the current timestamp of the block that this transaction is executed as in line 546.

SmartBaryFactory.sol

```
504 /// @notice Add a new LP to the pool. Can only be called by the owner.
505 /// DO NOT add the same LP token more than once.
506 function add(
507     IERC20[] memory _rewardTokens,
508     uint256[] memory _rewardMultipliers,
509     uint256 _rewardsStartTime,
510     uint256 _rewardPerSeconds,
511     uint256 _rewardsExpiration,
512     IERC20 _lpToken
513 ) internal {
514     require(
515         listAddedLPs[address(_lpToken)] == false,
516         "SmartBaryFactory: LP token already added"
517     );
518 
519     bytes memory bytecode = type(SmartBaryFactoryRewarder).creationCode;
520     bytecode = abi.encodePacked(bytecode, abi.encode(address(this)));
521     bytes32 salt = keccak256(abi.encodePacked(_lpToken, _rewardsStartTime));
522     address baryonFarmRewarder;
523 
524     assembly {
525         baryonFarmRewarder := create2(
526             0,
527             add(bytecode, 32),
528             mload(bytecode),
```

```
529             salt
530         )
531     }
532
533     SmartBaryFactoryRewarder(baryonFarmRewarder).initialize(
534         _rewardTokens,
535         _rewardMultipliers
536     );
537
538     lpToken.push(_lpToken);
539     rewarder.push(SmartBaryFactoryRewarder(baryonFarmRewarder));
540
541     poolInfo.push(
542         PoolInfo({
543             rewardsStartTime: _rewardsStartTime,
544             rewardsExpiration: _rewardsExpiration,
545             rewardPerSeconds: _rewardPerSeconds,
546             lastRewardTime: block.timestamp,
547             accRewardPerShare: 0
548         })
549     );
550
551     listAddedLPs[address(_lpToken)] = true;
552     emit PoolAdded(
553         lpToken.length.sub(1),
554         _rewardsStartTime,
555         _rewardsExpiration,
556         _rewardPerSeconds,
557         _lpToken,
558         baryonFarmRewarder
559     );
560 }
```

In addition, the pool can be renewed again (re-activate the pool after it expires) through the internal `set()` function, called from the `setPool()` function.

SmartBaryFactory.sol

```
580 /// @notice Update the given pool's information
581 function set(
582     uint256 _pid,
583     uint256 _rewardsStartTime,
584     uint256 _rewardsExpiration,
585     uint256 _rewardPerSeconds,
586     SmartBaryFactoryRewarder _rewarder,
587     bool overwrite
588 ) internal {
589     poolInfo[_pid].rewardsStartTime = _rewardsStartTime;
```

```
590     poolInfo[_pid].rewardsExpiration = _rewardsExpiration;
591     poolInfo[_pid].rewardPerSeconds = _rewardPerSeconds;
592     if (overwrite) {
593         rewarder[_pid] = _rewarder;
594     }
595     emit PoolSet(
596         _pid,
597         _rewardsStartTime,
598         _rewardsExpiration,
599         _rewardPerSeconds,
600         overwrite ? _rewarder : rewarder[_pid],
601         overwrite
602     );
603 }
```

The reward miscalculation will occur when the set time of `rewardsStartTime` is far away from the `lastRewardTime`. This is because when the reward time of the added pool or set pool starts and gets updated through the `updatePool()` function for the first time, it will apply the `pool.lastRewardTime` as the factor to measure the time passed from the last update of `pool.accRewardPerShare` which was set from the added block timestamp instead of the start block timestamp of reward distribution for the `add()` function. And for the `set()` function, it will apply the `rewardsExpiration` timestamp of the previous configuration as shown in line 668.

SmartBaryFactory.sol

```
656 /// @notice Update reward variables of the given pool.
657 /// @param pid The index of the pool.
658 /// @return pool Returns the pool updated.
659 function updatePool(uint256 pid) public returns (PoolInfo memory pool) {
660     pool = poolInfo[pid];
661     if (
662         block.timestamp > pool.lastRewardTime &&
663         block.timestamp > pool.rewardsStartTime
664     ) {
665         uint256 lpSupply = lpToken[pid].balanceOf(address(this));
666         if (lpSupply > 0) {
667             uint256 time = block.timestamp <= pool.rewardsExpiration
668                 ? block.timestamp.sub(pool.lastRewardTime) // Accrue rewards
    until now
669                 : pool.rewardsExpiration > pool.lastRewardTime
670                 ? pool.rewardsExpiration.sub(pool.lastRewardTime) // Accrue
    rewards until expiration
671                 : 0; // No rewards to accrue
672             uint256 reward = time.mul(pool.rewardPerSeconds);
673             pool.accRewardPerShare = pool.accRewardPerShare.add(
674                 (reward.mul(ACC_REWARD_PRECISION) / lpSupply)
675             );
```

```
676         }
677         pool.lastRewardTime = block.timestamp;
678         poolInfo[pid] = pool;
679         emit PoolUpdate(
680             pid,
681             pool.lastRewardTime,
682             lpSupply,
683             pool.accRewardPerShare
684         );
685     }
686 }
```

Therefore, the time passed calculation for adding a pool will be `block.timestamp - block.timestamp` (when the pool is added) instead of `block.timestamp - pool.rewardsStartTime`, and the time passed calculation for renewing a pool will be `block.timestamp - pool.expirationTime` (before renew this pool) instead of `block.timestamp - pool.rewardsStartTime` (after renew this pool).

This can offer the reward amount greater than what it should be for the users due to the fact that the `pool.accRewardPerShare` is inflated since it was accumulated when the pool was being added.

5.2.2. Remediation

Inspex suggests adding a condition to assign a value of `pool.lastRewardTime` as the same as the `pool.rewardStartTime` or the current timestamp if it is greater than the `pool.rewardStartTime` (in case the platform owner wants to add a pool with `pool.rewardStartTime` as the past) as in line 541.

SmartBaryFactory.sol

```
504 /// @notice Add a new LP to the pool. Can only be called by the owner.
505 /// DO NOT add the same LP token more than once.
506 function add(
507     IERC20[] memory _rewardTokens,
508     uint256[] memory _rewardMultipliers,
509     uint256 _rewardsStartTime,
510     uint256 _rewardPerSeconds,
511     uint256 _rewardsExpiration,
512     IERC20 _lpToken
513 ) internal {
514     require(
515         listAddedLPs[address(_lpToken)] == false,
516         "SmartBaryFactory: LP token already added"
517     );
518 
519     bytes memory bytecode = type(SmartBaryFactoryRewarder).creationCode;
520     bytecode = abi.encodePacked(bytecode, abi.encode(address(this)));
521     bytes32 salt = keccak256(abi.encodePacked(_lpToken, _rewardsStartTime));
522     address baryonFarmRewarder;
```

```
523
524     assembly {
525         baryonFarmRewarder := create2(
526             0,
527             add(bytecode, 32),
528             mload(bytecode),
529             salt
530         )
531     }
532
533     SmartBaryFactoryRewarder(baryonFarmRewarder).initialize(
534         _rewardTokens,
535         _rewardMultipliers
536     );
537
538     lpToken.push(_lpToken);
539     rewarder.push(SmartBaryFactoryRewarder(baryonFarmRewarder));
540
541     uint256 lastRewardTime = block.timestamp >= _rewardsStartTime ?
    block.timestamp: _rewardsStartTime
542     poolInfo.push(
543         PoolInfo({
544             rewardsStartTime: _rewardsStartTime,
545             rewardsExpiration: _rewardsExpiration,
546             rewardPerSeconds: _rewardPerSeconds,
547             lastRewardTime: lastRewardTime,
548             accRewardPerShare: 0
549         })
550     );
551
552     listAddedLPs[address(_lpToken)] = true;
553     emit PoolAdded(
554         lpToken.length.sub(1),
555         _rewardsStartTime,
556         _rewardsExpiration,
557         _rewardPerSeconds,
558         _lpToken,
559         baryonFarmRewarder
560     );
561 }
```

And for renewing a pool:

SmartBaryFactory.sol

```
580 /// @notice Update the given pool's information
581 function set(
582     uint256 _pid,
583     uint256 _rewardsStartTime,
584     uint256 _rewardsExpiration,
585     uint256 _rewardPerSeconds,
586     SmartBaryFactoryRewarder _rewarder,
587     bool overwrite
588 ) internal {
589     require(
590             _rewardsExpiration > _rewardsStartTime,
591             "SmartBaryFactory: Invalid time"
592     );
593     require(
594             _rewardsExpiration > block.timestamp,
595             "SmartBaryFactory: Invalid expiration time"
596     );
597     poolInfo[_pid].rewardsStartTime = _rewardsStartTime;
598     poolInfo[_pid].rewardsExpiration = _rewardsExpiration;
599     poolInfo[_pid].rewardPerSeconds = _rewardPerSeconds;
600     poolInfo[_pid].lastRewardTime = _rewardsStartTime;
601     if (overwrite) {
602         rewarder[_pid] = _rewarder;
603     }
604     emit PoolSet(
605         _pid,
606         _rewardsStartTime,
607         _rewardsExpiration,
608         _rewardPerSeconds,
609         overwrite ? _rewarder : rewarder[_pid],
610         overwrite
611     );
612 }
```

5.3. Design Flaw in Reward Distribution Mechanism

ID	IDX-003
Target	SmartBaryFactory SmartBaryFactoryRewarder
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Low **Impact:** Medium The platform users will not be able to claim the reward when the balance in the pool reward is insufficient. However, they can withdraw all their stake tokens anytime if they want to. **Likelihood:** Low It is unlikely that the platform owner will not transfer the reward to the pool constantly since this will cause reputation damage to the platform.
Status	Resolved Baryon Network team has resolved this issue as suggested by adjusting the reward distribution mechanism by transferring all the required tokens when the pool is added or set to ensure that the reward is sufficient in commit `426be5e620883e5a06a15fc97501323f1ee77750`.

5.3.1. Description

In the ideal case, the platform owner should transfer the reward tokens to the reward contract sufficiently before establishing the new pool. However, with the current design, if the reward amount is greater than the balance in the reward contract, the difference will be kept in the `rewardDebt` state and the users can withdraw it later. Therefore, the users might not be able to claim the reward immediately until the platform owner transfers reward tokens to the reward contract.

5.3.2. Remediation

Inspex suggests adjusting the reward distribution mechanism by transferring all the required tokens when the pool is added or set to ensure that the reward is sufficient for distributing to the users when they claim it. Basically, this can be calculated by applying these pseudocodes to the `SmartBaryFactory` contract:

Add a pool (`add()` function):

```
require(newExpiration > newRewardsStartTime);
require(newExpiration > block.timestamp);

uint256 rewardAmountWith1x = (_rewardsExpiration - _rewardsStartTime) *
```

```
    _rewardPerSeconds;
5   for (uint256 i=0; i<_rewardTokens.length; i++) {
6       _rewardTokens[i].safeTransferFrom(msg.sender, baryonFarmRewarder,
    rewardAmountWith1x * _rewardMultipliers[i]);
7       claimedAmount[i] = 0;
8       oldReserveBalance[i] = 0;
9   }
```

Set a pool (`set()` function):

```
1   require(_rewardsExpiration > _rewardsStartTime);
2   require(_rewardsExpiration > block.timestamp);
3
4   // force _rewardsStartTime to be block.timestamp as minimum this helps select
    the case of continue the pool and resurrect the inactive pool
5   uint256 _rewardsStartTime = _rewardsStartTime > block.timestamp?
    _rewardsStartTime: block.timestamp;
6
7
8   uint256 oldExpiration = block.timestamp > pool.rewardsExpiration ?
    pool.rewardsExpiration : block.timestamp;
9   uint256 newRewardAmountWith1x = (_rewardsExpiration - _rewardsStartTime) *
    _rewardPerSeconds;
10
11  PoolInfo storage pool = poolInfo[_pid];
12  SmartBaryFactoryRewarder factoryRewarder = rewarder[_pid];
13  IERC20[] tokens = factoryRewarder.getRewardTokens();
14  uint256[] multipliers = factoryRewarder.getRewardMultipliers();
15
16  for(i=0, i<tokens.length, i++){
17      uint256 tokenBalance =
    IERC20(tokens[i]).balanceOf(address(factoryRewarder));
18      uint256 rewardMultiplier = multipliers[i];
19      uint256 oldReserveBalance = (oldExpiration - pool.rewardsStartTime) *
    pool.rewardPerSeconds + pool.oldReserveBalance[i] - pool.claimedAmount[i];
20      uint256 remainReserveReward = tokenBalance > oldReserveBalance?
    tokenBalance - oldReserveBalance: 0;
21      uint256 toTransferMore = newRewardAmountWith1x * rewardMultiplier >
    remainReserveReward? (newRewardAmountWith1x * rewardMultiplier) -
    remainReserveReward: 0;
22      IERC20(tokens[i]).safeTransferFrom(msg.sender, address(factoryRewarder),
    toTransferMore);
23      pool.oldReserveBalance[i] = oldReserveBalance;
24      pool.claimedAmount[i] = 0;
25  }
26
27  pool.rewardsStartTime = _rewardsStartTime;
28  pool.rewardsExpiration = _rewardsExpiration;
```

```
29 pool.rewardPerSeconds = _rewardPerSeconds;
```

`pool.claimedAmount[i]` can be updated when the user claims the reward through the `deposit()`, `harvest()` and `withdrawAndHarvest()` functions. This is used to track the reward amount of users who have claimed the reward for the current reward configuration. Hence, in the next reward configuration, this amount will be deducted with the previous reserve reward amount (`oldReserveBalance`) to find the optimal balance of the next deposit.

5.4. Insufficient Logging for Privileged Functions

ID	IDX-004
Target	SmartBaryFactory
Category	General Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Very Low **Impact:** Low Privileged functions' executions cannot be monitored easily by the users. **Likelihood:** Low It is not likely that the execution of the privileged functions will be a malicious action.
Status	Resolved Baryon Network team has resolved this issue as suggested by emitting events for the execution of privileged functions in commit `426be5e620883e5a06a15fc97501323f1ee77750`.

5.4.1. Description

Privileged functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions, allowing the controlling parties to perform actions that cause big impacts on the platform.

For example, the owner can set the reward tokens by executing the `withdrawMultiplePool()` function in the `SmartBaryFactory` contract, and no events are emitted.

SmartBaryFactory.sol

```
790 /// @notice Withdraw all token reward in pool
791 /// @param pid The index of the pool.
792 function withdrawMultiplePool(uint256[] calldata pid) public onlyOwner {
793     for (uint256 i = 0; i < pid.length; i++) {
794         SmartBaryFactoryRewarder _rewarder = rewarder[pid[i]];
795         if (address(_rewarder) != address(0)) {
796             _rewarder.withdrawForOwner();
797         }
798     }
799 }
```

The privileged functions without sufficient logging are as follows:

File	Contract	Function
SmartBaryFactory.sol (L:792)	SmartBaryFactory	withdrawMultiplePool()
SmartBaryFactory.sol (L:804)	SmartBaryFactory	withdrawPoolTokens()
SmartBaryFactory.sol (L:818)	SmartBaryFactory	setRewardConfig()
SmartBaryFactory.sol (L:831)	SmartBaryFactory	withdrawMultiple()

5.4.2. Remediation

Inspex suggests emitting events for the execution of privileged functions, for example:

SmartBaryFactory.sol

```
789 event WithdrawMultiplePool(uint256[] calldata pid);
790 /// @notice Withdraw all token reward in pool
791 /// @param pid The index of the pool.
792 function withdrawMultiplePool(uint256[] calldata pid) public onlyOwner {
793     for (uint256 i = 0; i < pid.length; i++) {
794         SmartBaryFactoryRewarder _rewarder = rewarder[pid[i]];
795         if (address(_rewarder) != address(0)) {
796             _rewarder.withdrawForOwner();
797         }
798     }
799     emit WithdrawMultiplePool(pid);
800 }
```

5.5. Improper Function Visibility

ID	IDX-005
Target	SmartBaryFactory SmartBaryFactoryRewarder
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved Baryon Network has resolved this issue as suggested by changing the functions' visibility to external in commit `426be5e620883e5a06a15fc97501323f1ee77750`.

5.5.1. Description

Public functions that are never called internally by the contract itself should have external visibility. This improves the readability of the contract, allowing clear distinction between functions that are externally used and functions that are also called internally.

The following source code shows that the `setRewardConfig()` function visibility in the `SmartBaryFactory` contact is set to public and it is never called from any internal function.

SmartBaryFactory.sol

```
818 function setRewardConfig(
819     uint256 pid,
820     IERC20[] memory _rewardTokens,
821     uint256[] memory _rewardMultipliers
822 ) public onlyOwner {
823     SmartBaryFactoryRewarder _rewarder = rewarder[pid];
824     if (address(_rewarder) != address(0)) {
825         _rewarder.initialize(_rewardTokens, _rewardMultipliers);
826     }
827 }
```

The following table contains all functions that have public visibility and are never called from any internal function.

File	Contract	Function
SmartBaryFactory.sol (L:249)	SmartBaryFactoryRewarder	initialize()
SmartBaryFactory.sol (L:278)	SmartBaryFactoryRewarder	claimReward()
SmartBaryFactory.sol (L:334)	SmartBaryFactoryRewarder	withdrawForOwner()
SmartBaryFactory.sol (L:345)	SmartBaryFactoryRewarder	withdrawTokens()
SmartBaryFactory.sol (L:792)	SmartBaryFactory	withdrawMultiplePool()
SmartBaryFactory.sol (L:804)	SmartBaryFactory	withdrawPoolTokens()
SmartBaryFactory.sol (L:818)	SmartBaryFactory	setRewardConfig()
SmartBaryFactory.sol (L:831)	SmartBaryFactory	withdrawMultiple()

5.5.2. Remediation

Inspex suggests changing all functions' visibility to external if they are not called from any internal function as shown in the following example:

SmartBaryFactory.sol

```
818 function setRewardConfig(
819     uint256 pid,
820     IERC20[] memory _rewardTokens,
821     uint256[] memory _rewardMultipliers
822 ) external onlyOwner {
823     SmartBaryFactoryRewarder _rewarder = rewarder[pid];
824     if (address(_rewarder) != address(0)) {
825         _rewarder.initialize(_rewardTokens, _rewardMultipliers);
826     }
827 }
```

5.6. Inexplicit Solidity Compiler Version

ID	IDX-006
Target	SmartBaryFactory SmartBaryFactoryRewarder
Category	Smart Contract Best Practice
CWE	Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved Baryon Network team has resolved this issue as suggested by fixing the Solidity compiler to the latest stable version in commit `426be5e620883e5a06a15fc97501323f1ee77750`.

5.6.1. Description

The Solidity compiler versions declared in the smart contracts were not explicit. Each compilation may be done using different compiler versions, which may potentially result in compatibility issues.

SmartBaryFactory.sol

```
3 pragma solidity >=0.8.0;
```

The following table contains all contracts which the inexplicit compiler declares.

Contract	Version
SmartBaryFactory	>=0.8.0
SmartBaryFactoryRewarder	>=0.8.0

5.6.2. Remediation

Inspex suggests fixing the Solidity compiler to the latest stable version. At the time of the audit, the latest stable version of Solidity compiler in major 0.8 is v0.8.13 (https://github.com/ethereum/solidity/releases/tag/v0.8.13)

SmartBaryFactory.sol

```
3 pragma solidity 0.8.13;
```

5.7. Unsupported Design for Deflationary Tokens

ID	IDX-007
Target	SmartBaryFactory
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved Baryon Network team has resolved this issue as suggested by modifying the logic of the `deposit()` function to validate the amount of the received token from the user instead of using the value of `amount` parameter directly in commit `426be5e620883e5a06a15fc97501323f1ee77750`.

5.7.1. Description

In the `SmartBaryFactory` contract, the users can deposit their tokens to acquire rewards. The deposited tokens can be a simple ERC20 token or LP token depending on the pools added by the contract owner.

However, in the `deposit()` function, an issue could arise when the pool uses a deflationary token (the token that reduces the circulating supply itself when it is transferred).

This means the `amount` that the user deposits will be reduced due to the deflationary mechanism, but the contract recognizes it as the full amount as in line 712.

SmartBaryFactory.sol

```
688 /// @notice Deposit LP tokens to Pool for reward allocation.
689 /// @param pid The index of the pool.
690 /// @param amount LP token amount to deposit.
691 function deposit(uint256 pid, uint256 amount) public {
692     PoolInfo memory pool = updatePool(pid);
693     farmCheckStartTime(pool);
694     farmCheckExpirationTime(pool);
695
696     address to = msg.sender;
697
698     UserInfo storage user = userInfo[pid][to];
699     lpToken[pid].safeTransferFrom(to, address(this), amount);
700
701     uint256 accumulatedReward = uint256(
```

```
702             user.amount.mul(pool.accRewardPerShare) / ACC_REWARD_PRECISION
703         );
704         uint256 _pendingReward = accumulatedReward.sub(user.rewardDebt)
705 
706         SmartBaryFactoryRewarder _rewarder = rewarder[pid];
707         if (address(_rewarder) != address(0)) {
708             _rewarder.claimReward(to, _pendingReward);
709         }
710 
711         // Updated information
712         user.amount = user.amount.add(amount);
713         user.rewardDebt = accumulatedReward.add(
714             uint256(amount.mul(pool.accRewardPerShare) / ACC_REWARD_PRECISION)
715         );
716 
717         emit Deposit(to, pid, amount, to);
718     }
```

The failure of recognizing the token amount could lead to the following scenarios:

Scenario 1: Unable to withdraw staking tokens

Assuming that there is a pool in the `SmartBaryFactory` contract which receives a deflationary token ($TOKEN) with 10% burn rate when the token is transferred.

Currently, there is only User A who stakes $TOKEN to the $TOKEN pool in the `SmartBaryFactory` contract.

Holder	Balance
User A	100

Total $TOKEN in the `SmartBaryFactory` contract: 90

User B deposits 100 $TOKEN to the $TOKEN pool in the `SmartBaryFactory` contract. The `SmartBaryFactory` contract will receive 90 $TOKEN since $TOKEN is 10% deduction from the deflationary mechanism, in this case 10 $TOKEN.

Holder	Balance
User A	100
User B	100

Total $TOKEN in the `SmartBaryFactory` contract: 180

User B then withdraws 100 $TOKEN from the `SmartBaryFactory` contract. The `SmartBaryFactory` contract will validate whether the withdrawn `amount` exceeds the `user.amount` or not as in line 763 due to the `SafeMath.sub()`.

SmartBaryFactory.sol

```
745 /// @notice Withdraw LP tokens from Factory and harvest reward for deposited
    user.
746 /// @param pid The index of the pool.
747 /// @param amount LP token amount to withdraw.
748 function withdrawAndHarvest(uint256 pid, uint256 amount) public {
749     PoolInfo memory pool = updatePool(pid);
750     farmCheckStartTime(pool);
751     address to = msg.sender;
752
753     UserInfo storage user = userInfo[pid][to];
754     uint256 accumulatedReward = uint256(
755         user.amount.mul(pool.accRewardPerShare) / ACC_REWARD_PRECISION
756     );
757     uint256 _pendingReward = accumulatedReward.sub(user.rewardDebt);
758
759     // Updated information
760     user.rewardDebt = accumulatedReward.sub(
761         uint256(amount.mul(pool.accRewardPerShare) / ACC_REWARD_PRECISION)
762     );
763     user.amount = user.amount.sub(amount);
764
765     SmartBaryFactoryRewarder _rewarder = rewarder[pid];
766     if (address(_rewarder) != address(0)) {
767         _rewarder.claimReward(to, _pendingReward);
768     }
769
770     lpToken[pid].safeTransfer(to, amount);
771
772     emit Withdraw(to, pid, amount, to);
773     emit Harvest(to, pid, _pendingReward);
774 }
```

Since User B deposited 100 $TOKEN and the balance of $TOKEN in the contract is greater than 100, User B is allowed to withdraw 100 $TOKEN.

Holder	Balance
User A	100
User B	0

Total $TOKEN in the `SmartBaryFactory` contract: 80

As a result, if User A decides to withdraw 100 $TOKEN, this transaction will be reverted since the balance in the contract is insufficient due to the `lpToken[pid].safeTransfer(to, amount)` in line 770.

Scenario 2: Reward Calculation Exploit

By repeating the `Scenario 1: Unable to withdraw staking tokens`, this also means reducing the `lpToken` which affects to the reward calculation as in line `673`.

SmartBaryFactory.sol

```
656 /// @notice Update reward variables of the given pool.
657 /// @param pid The index of the pool.
658 /// @return pool Returns the pool updated.
659 function updatePool(uint256 pid) public returns (PoolInfo memory pool) {
660     pool = poolInfo[pid];
661     if (
662         block.timestamp > pool.lastRewardTime &&
663         block.timestamp > pool.rewardsStartTime
664     ) {
665         uint256 lpSupply = lpToken[pid].balanceOf(address(this));
666         if (lpSupply > 0) {
667             uint256 time = block.timestamp <= pool.rewardsExpiration
668                 ? block.timestamp.sub(pool.lastRewardTime) // Accrue rewards
    until now
669                 : pool.rewardsExpiration > pool.lastRewardTime
670                 ? pool.rewardsExpiration.sub(pool.lastRewardTime) // Accrue
    rewards until expiration
671                 : 0; // No rewards to accrue
672             uint256 reward = time.mul(pool.rewardPerSeconds);
673             pool.accRewardPerShare = pool.accRewardPerShare.add(
674                 (reward.mul(ACC_REWARD_PRECISION) / lpSupply)
675             );
676         }
677         pool.lastRewardTime = block.timestamp;
678         poolInfo[pid] = pool;
679         emit PoolUpdate(
680             pid,
681             pool.lastRewardTime,
682             lpSupply,
683             pool.accRewardPerShare
684         );
685     }
686 }
```

Therefore, an attacker can execute the `Scenario 1: Unable to withdraw staking tokens` repeatedly to gain a large amount of reward by amplifying the `pool.accRewardPerShare`.

5.7.2. Remediation

Inspex suggests modifying the logic of the `deposit()` function to validate the amount of the received token from the user instead of using the value of `amount` parameter directly as in line 699 - 702 and 717, or not allowing the deflationary token as the `lpToken` at all.

SmartBaryFactory.sol

```
688 /// @notice Deposit LP tokens to Pool for reward allocation.
689 /// @param pid The index of the pool.
690 /// @param amount LP token amount to deposit.
691 function deposit(uint256 pid, uint256 amount) public {
692     PoolInfo memory pool = updatePool(pid);
693     farmCheckStartTime(pool);
694     farmCheckExpirationTime(pool);
695 
696     address to = msg.sender;
697 
698     UserInfo storage user = userInfo[pid][to];
699     uint256 currentBal = lpToken[pid].balanceOf(address(this));
700     lpToken[pid].safeTransferFrom(to, address(this), amount);
701     uint256 afterBal = lpToken[pid].balanceOf(address(this));
702     uint256 realAmount = afterBal.sub(currentBal);
703 
704     uint256 accumulatedReward = uint256(
705         user.amount.mul(pool.accRewardPerShare) / ACC_REWARD_PRECISION
706     );
707     uint256 _pendingReward = accumulatedReward.sub(user.rewardDebt)
708 
709     SmartBaryFactoryRewarder _rewarder = rewarder[pid];
710     if (address(_rewarder) != address(0)) {
711         _rewarder.claimReward(to, _pendingReward);
712     }
713 
714     // Updated information
715     user.amount = user.amount.add(realAmount);
716     user.rewardDebt = accumulatedReward.add(
717         uint256(realAmount.mul(pool.accRewardPerShare) / ACC_REWARD_PRECISION)
718     );
719 
720     emit Deposit(to, pid, realAmount, to);
721 }
```

6. Appendix

6.1. About Inspex

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

inspeX

CYBERSECURITY PROFESSIONAL SERVICE