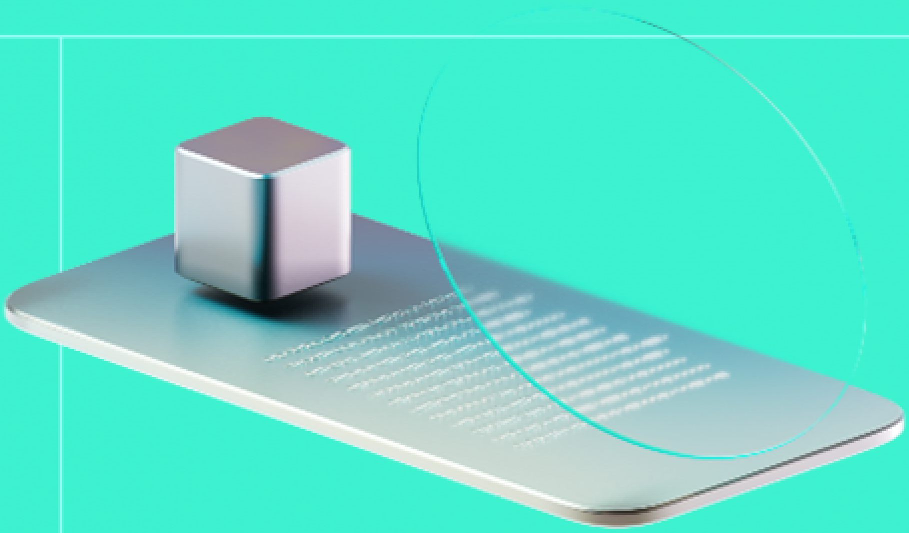
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Genie Swap

Date: 08/01/2024

We thank Genie Swap for allowing us to conduct a Smart Contract Security Assessment. This document outlines our methodology, limitations, and results of the security assessment.

GenieSwap is a farming protocol designed to allow users to open farms by locking in a specific farm token for a defined period.

Platform: EVM

Language: Solidity

Tags: Farming,Staking

Timeline: 26.12.2023 - 08.01.2024

Methodology: https://hackenio.cc/sc_methodology

Last Review Scope

Repository	https://github.com/Genieswap-com/simplified-farms-contracts
Commit	2fed0fdde983b6ec96592ec4d003cc33fd71540b

Audit Summary

10/10	10/10	100%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 10/10

The system users should acknowledge all the risks summed up in the risks section of the report

2	2	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	1
High	1
Medium	0
Low	0

Vulnerability	Status
F-2023-0286 - Exploitable "extendLockTime" Function in Farming Contract	Fixed
F-2023-0287 - Exploitable Parameter Adjustment via "extendLockTime" in Yield Calculation	Fixed

Document

Name	Smart Contract Code Review and Security Analysis Report for Genie Swap
Audited By	Kaan Caglan
Approved By	Przemyslaw Swiatowiec
Website	https://genieswap.com/
Changelog	27/12/2023 - Preliminary Report

Table to Contents

System Overview **6**
Privileged Roles 6
Executive Summary **7**
Documentation Quality 7
Code Quality 7
Test Coverage 7
Security Score 7
Summary 7
Risks **8**
Findings **9**
Vulnerability Details 9
Observation Details 14
F-2023-0279 - Floating Pragma - Info 14
F-2023-0280 - Revert String Size Optimization - Info 15
F-2023-0281 - Custom Errors In Solidity For Gas Efficiency - Info 18
F-2023-0282 - Storage Layout Optimization - Info 19
F-2023-0283 - Avoid Using State Variables Directly In `Emit` For Gas Efficiency - Info 20
F-2023-0284 - Owner Can Renounce Ownership - Info 21
F-2023-0285 - Inefficient Resource Usage In ResetFarm Function - Info 22
F-2023-0308 - Cache State Variables - Info 24
Disclaimers 25
Hacken Disclaimer 25
Technical Disclaimer 25
Appendix 1. Severity Definitions 26
Appendix 2. Scope 27

System Overview

GenieSwap is a farming protocol designed to allow users to open farms by locking in a specific farm token (and optionally a platform token for boosts) for a defined period. It facilitates earning yields based on the lock time and amount, with an option to boost returns. The system consists of two main contracts: **TimelockedFarm** for managing individual farms and **FarmFactory** for deploying these farm contracts.

TimeLockedFarm - Manages the lifecycle of individual farms where users can lock tokens to earn yield. It allows for operations like opening, boosting, extending, and ending farms.

FarmFactory - Deploys new instances of the **TimelockedFarm** contract with unique configurations. It maintains a registry mapping farm tokens to their respective farm contracts.

Privileged roles

- The owner of the contract can set parameters like minimum/maximum lock times, yield rates, and amounts.
- The owner of the contract can pause the contract or withdraw remaining tokens.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.
- Natspec is sufficient.

Code quality

The total Code Quality score is **10** out of **10**.

- The code follows the Solidity best practices.
- The code is structured and readable.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- All of the functionalities are covered by unit tests.

Security score

Upon auditing, it was found that the code contained **1** critical, **1** high, **0** medium, and **0** low severity issues. All the issues mentioned in the report were resolved, resulting in a security score of **10** out of **10**.

All identified issues are detailed in the "Findings" section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **10**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- No risks have been identified.

Findings

Vulnerability Details

F-2023-0286 - Exploitable ”extendLockTime” Function in Farming Contract - Critical

Description:	The `extendLockTime` function in the farming contract allows users to extend the lock time of their existing farms, recalculating and potentially increasing the yield based on the new lock time. However, a design flaw permits a user to initially stake tokens for a minimal duration, then extend the lock time just before withdrawing, thus earning a disproportionate yield. This issue circumvents the intended restriction of the lock-in period, allowing users to claim higher rewards without committing their tokens for the full duration.
Assets:	• contracts/Farm.sol [https://github.com/Genieswap-com/simplified-farms-contracts]
Status:	Fixed

Classification

Severity:	Critical
Impact:	5/5
Likelihood:	5/5

Recommendations

Recommendation: The `extendLockTime` function should also reset lock time as it is done in `boostFarm` and `topUpFarm`.

```
farm.startTime = block.timestamp;
```

Remediation (Revised commit: 926a0da): Client fixed this issue by adding a control to the `extendLockTime` function which ensures that extension can only be done within the original lock period.

```
// This check ensures that the extension can only be done within the original
// lock period, making users truly commit to the entire duration.
if (block.timestamp > startTime + oldLockTime) {
    revert FarmHasEnded();
}
```

Evidences

POC

Reproduce:

Proof of Concept (POC) Steps:

Initial Farm Opening by Alice:

- Alice opens a farm with **x** amount of tokens for the minimum allowed duration, say 1 day, to minimize her commitment.

Waiting Period:

- Alice waits for a significant period, e.g., 50 days, without interacting with her farm. During this time, her farm accrues no rewards due to the short initial lock time.

Strategic Extension of Lock Time:

- Alice contemplates ending her farm. If she had initially opened the farm with the intended 100-day lock period, she would be bound to either wait out the full term to claim rewards or use the `emergencyEndFarm` function prematurely to recover only her principal without any rewards. To avoid this restriction, she instead initially commits to a minimal 1-day period. Near the 50th day, realizing she wants to end her farm, Alice strategically calls `extendLockTime` with a new lock time set to 49 days, which is just under the actual time she has had her tokens staked but far less than the 100 days she initially planned. This maneuver allows her to immediately use `endFarm` to claim a reward for 50 days, exploiting the system to obtain rewards she would not have been entitled to under a straightforward 100-day commitment.

Immediate Ending of Farm:

- Immediately after extending the lock time, Alice calls `**endFarm**`. Since the actual time passed (50 days) is now greater than the newly set lock time (49 days + 1 days initially), she successfully ends her farm and claims rewards for 50 days, despite initially committing to only 1 day.

Proof of Concept (POC) :

```
>>> farm.openFarm(ONE_DAY, 10e18, False, {'from': alice})
Transaction sent: 0xc3ce4e951430a0987d5e
```

See more

F-2023-0287 - Exploitable Parameter Adjustment via ”extendLockTime” in Yield Calculation - High

Description: The yield calculation mechanism within the farming contract is designed to reward users based on a set of parameters, including a maximum percentage rate (`maxPr`). However, a vulnerability exists where users can exploit the `extendLockTime` function to benefit from updated maximum percentage rates (`maxPr`) or other beneficial parameter changes made after their farm was initially opened. This issue arises because the `extendLockTime` function recalculates the yield based on the current parameters, disregarding that these changes were intended only to affect new farms opened after the changes.

Assets:

- contracts/Farm.sol [https://github.com/Genieswap-com/simplified-farms-contracts]

Status: Fixed

Classification

Severity: High

Impact: 5/5

Likelihood: 3/5

Recommendations

Recommendation: It is recommended to lock parameter values for active farms when a farm is opened. Snapshot the relevant parameters (e.g., `maxPr`, `minPr`) and ensure that the yield for this farm is calculated based on these locked-in values, even if parameters are later adjusted.

Remediation (Revised commit: 926a0da): Client fixed this issue by adding a snapshot to those relevant parameters in `openFarm` function and using them when calculating the reward.

```
// Snapshot parameters
farm.minPr = minPr;
farm.maxPr = maxPr;
farm.minLockTime = minLockTime;
farm.maxLockTime = maxLockTime;
farm.boostFactor = boostFactor;
```

Evidences

POC

Reproduce:

Proof of Concept (POC) Steps :

Initial Setup by Protocol:

- The protocol sets `maxLockTime` to 100 days and `maxPr` to 5000, meaning users staking for 100 days can earn a reward of up to 50% of their staked amount.

Alice's Strategic Farm Opening:

- Alice opens a farm with a lock time of 100 days minus 1 second, allowing her the flexibility to call `extendLockTime` later. She stakes her tokens under the initial `maxPr` of 5000.

Protocol Parameter Adjustment:

- 50 days after Alice's farm opening, the protocol decides to incentivize longer staking periods by increasing `maxPr` to 7500, permitting future users to earn up to 75% of their staked amount over 100 days.

Alice's Exploitation of Parameter Change:

- Seeing an opportunity, Alice calls `extendLockTime` with exactly 100 days, just before her original term ends. This action triggers a recalculation of her yield based on the new `maxPr` of 7500. As a result, Alice is positioned to receive a 75% reward, significantly more than her initial potential reward, and also more than what was intended for users who staked under the old terms.

Proof of Concept (POC) :

```
>>> farm.openFarm((100*ONE_DAY)-1, 10e18, False, {'from': alice})
Transaction sent: 0x0d9d51a2a782b78bfdb984f47c86f93b4d96ec89df45f955d4f52e989745a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  TimelockedFarm.openFarm confirmed   Block: 10   Gas used: 170745 (1.42%)

<Transaction '0x0d9d51a2a782b78bfdb984f47c86f93b4d96ec89df45f955d4f52e989745a4df
>>> farm.farms(alice)
(1703372451, 8639999, 4999000000000000000, True, False)
>>> chain.sleep(50*ONE_DAY)
>>> farm.setMaxPr(7500, {'f
```

See more

Observation Details

F-2023-0279 - Floating Pragma - Info

Description: The project uses floating pragma `^0.8.20;`

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Assets:

- contracts/Farm.sol [https://github.com/Genieswap-com/simplified-farms-contracts]

Status: Fixed

Recommendations

Recommendation: It is recommended to lock pragma version.

Remediation (Revised commit: 29d3699): Client locked the pragma version.

F-2023-0280 - Revert String Size Optimization - Info

Description:

Shortening the revert strings to fit within 32 bytes will decrease deployment time and decrease runtime Gas when the revert condition is met.

Revert strings that are longer than 32 bytes require at least one additional `mstore`, along with additional overhead to calculate memory offset.

```
Path: ./contracts/Farm.sol

    require(
        config.farmToken != address(0),
        "farm token address must not be zero"
    );

    require(farmOwner != address(0), "farm owner must not be address zer

    require(
        _farmToken.balanceOf(msg.sender) >= amount,
        "farmer does not have enough farm-tokens"
    );

        require(
            _platformToken.allowance(msg.sender, address(this)) >=
                boostAmount,
            "allowance for platform-token too low"
        );

        require(
            _platformToken.balanceOf(msg.sender) >= boostAmount,
            "farmer does not have enough platform-tokens"
        );

    require(
        _farmToken.balanceOf(address(this)) >= totalTokensToPayout,
        "farm does not have enough tokens to pay out yield"
    );

    require(
        _farmToken.balanceOf(msg.sender) >= amount,
        "farmer does not have enough farm-tokens"
    );

    require(
        _farmToken.balanceOf(address(this)) >= totalTokensToPayout,
        "farm does not have enough tokens to pay out yield"
        );

        require(
            _platformToken.allowance(msg.sender, address(this)) >= boostAmou
            "allowance for platform-token too low"
        );

        require(
            _platformToken.balanceOf(msg.sender) >= boostAmount,
            "farmer does not have enough platform-tokens"
        );

        require(
            _farmToken.balanceOf(address(this)) >= totalTokensToPayout,
            "farm does not have enough tokens to pay out yield"
        );

        require(
            _farmToken.balanceOf(address(this)) >= totalTokensToPayout,
            "farm does not have enough tokens to pay out yield"
        );

        require(
            block.timestamp <= farm.startTime + farm.lockTime,
            "farm has ended, end farm normally"
        );

        require(
            minLockTime_ >= 86400,
            "minumum lock time must be greater equal than one day"
        );

        require(minPr_ >= 1, "minumum PR must be greater equal than 1");

        require(
            minAmount_ < maxAmount,
            "invalid new minimal farm token amount"
        );

        require(minAmount_ >= 1, "minumum amount must be greater equal than

        require(
            maxAmount_ > minAmount,
            "invalid new maximum farm token amount"
        );

            require(
                address(_platformToken) != address(0),
                "platform token address must not be zero to enable boost"
            );

            require(
                boostFactor_ > BASIS,
                "cannot enable boost when boost factor is less than 1"
            );
```

Assets:

- contracts/Farm.sol [https://github.com/Genieswap-com/simplified-farms-contracts]

Status: Fixed

Recommendations

Recommendation: To optimize Gas usage in your Solidity contract, it is recommended to keep revert strings as short as possible and to ensure that they fit within 32 bytes. It is possible to use abbreviations or simplified error messages to keep the string length short. Doing so can reduce the amount of Gas used during deployment and runtime when the revert condition is met.

Remediation (Revised commit: 0400efb): Client removed require statements that use more than 32 bytes and introduced checks with custom errors.

F-2023-0281 - Custom Errors in Solidity for Gas Efficiency - Info

Description:

Starting from Solidity version 0.8.4, the language introduced a feature known as "custom errors". These custom errors provide a way for developers to define more descriptive and semantically meaningful error conditions without relying on string messages. Prior to this version, developers often used the `require` statement with string error messages to handle specific conditions or validations. However, every unique string used as a revert reason consumes gas, making transactions more expensive.

Custom errors, on the other hand, are identified by their name and the types of their parameters only, and they do not have the overhead of string storage. This means that, when using custom errors instead of `require` statements with string messages, the gas consumption can be significantly reduced, leading to more gas-efficient contracts.

Assets:

- contracts/Farm.sol [https://github.com/Genieswap-com/simplified-farms-contracts]

Status: Fixed

Recommendations

Recommendation:

It is recommended to use custom errors instead of revert strings to reduce gas costs, especially during contract deployment. Custom errors can be defined using the error keyword and can include dynamic information.

Remediation (Revised commit: 0400efb): Client removed require statements and introduced custom errors for functions that do not have `onlyOwner` modifier.

F-2023-0282 - Storage Layout Optimization - Info

Description:

Storage Layout Optimization in Solidity involves arranging state variables to minimize gas costs. Since storage is expensive, combining variables into as few slots as possible and deleting unneeded variables can significantly reduce the gas needed for contract operations.

The variable definition

```
bool public boostEnabled;
```

should be come after `IERC20Metadata private _platformToken;` to save 1 slot.

Assets:

- contracts/Farm.sol [https://github.com/Genieswap-com/simplified-farms-contracts]

Status: Fixed

Recommendations

Recommendation:

To optimize storage and reduce gas costs, rearrange the storage variables in a way that makes the most of each 32-byte storage slot.

Remediation (Revised commit: 6bd8102): Client optimized the storage by moving `boostEnabled` variable after an interface definition.

F-2023-0283 - Avoid Using State Variables Directly in `emit` for Gas Efficiency - Info

Description:

In Solidity, function visibility is an important aspect that determines how and where a function can be called from. Two commonly used visibilities are `public` and `external`. A `public` function can be called both from other functions inside the same contract and from outside transactions, while an `external` function can only be called from outside the contract.

A potential pitfall in smart contract development is the misuse of the `public` keyword for functions that are only meant to be accessed externally. When a function is not used internally within a contract and is only intended for external calls, it should be labeled as `external` rather than `public`. Using `public` unnecessarily can introduce potential vulnerabilities and also make the contract consume more gas than required. This is because `public` functions have to add additional code to handle both internal and external calls, while `external` functions can be more optimized since they only handle external calls.

```
emit MinLockTimeChanged(minLockTime);

emit MaxLockTimeChanged(maxLockTime);

emit MinPrChanged(minPr);

emit MaxPrChanged(maxPr);

emit MinAmountChanged(minAmount);

emit MaxAmountChanged(maxAmount);
```

Assets:

- contracts/Farm.sol [https://github.com/Genieswap-com/simplified-farms-contracts]

Status: Fixed

Recommendations

Recommendation: To reduce gas costs and maintain predictable contract behavior, consider using local variables to store state variable values before emitting events. This practice eliminates costly state variable lookups and ensures smoother contract execution.

Remediation (Revised commit: 3a2abf9): Client fixed this issue by using a stack variable in emits instead of a state variable.

F-2023-0284 - Owner Can Renounce Ownership - Info

Description:	The smart contract under inspection inherits from the `Ownable` library, which provides basic authorization control functions, simplifying the implementation of user permissions. While the contract allows for the transfer of ownership to a different address or account, it also retains the default `renounceOwnership` function from `Ownable`. Once the owner uses this function to renounce ownership, the contract becomes ownerless. Evidence in the transaction logs shows that, following the activation of the `renounceOwnership` function, any attempts to invoke functions requiring owner permissions fail, with the error message: `"Ownable: caller is not the owner."` This condition makes the contract's adjustable parameters immutable, potentially rendering the contract ineffective for any future administrative modifications that might be needed.
Assets:	• contracts/Farm.sol [https://github.com/Genieswap-com/simplified-farms-contracts]
Status:	Fixed

Recommendations

Recommendation: To mitigate this vulnerability:

- Override the `renounceOwnership` function to revert transactions: By overriding this function to simply revert any transaction, it will become impossible for the contract owner to unintentionally (or intentionally) render the contract ownerless and thus immutable.

Remediation (Revised commit: 9515b08): Client fixed this issue by overriding `renounceOwnership` function, which now reverts renounce ownership transactions.

F-2023-0285 - Inefficient Resource Usage in resetFarm Function - Info

Description: The `resetFarm` function is designed to reset the farming details of a given farmer by manually setting each variable in the `Farm` struct to its default value. This includes setting integer values to 0 and boolean values to false. However, this method of individually resetting each property is less efficient and potentially more error-prone compared to utilizing the Solidity `delete` keyword. The `delete` keyword resets a struct to its default values in a single operation, which not only simplifies the code but also potentially reduces gas costs by minimizing the number of operations required to reset the struct.

```
    function resetFarm(address farmer) private {
        Farm storage farm = farms[farmer];

        farm.lockedAmount[address(_farmToken)] = 0;
        farm.lockedAmount[address(_platformToken)] = 0;

        farm.active = false;
        farm.boosted = false;
        farm.lockTime = 0;
        farm.startTime = 0;
        farm.yield = 0;
    }
```

Assets:

- contracts/Farm.sol [https://github.com/Genieswap-com/simplified-farms-contracts]

Status: Fixed

Recommendations

Recommendation: Replace the individual assignments of variables with a single `delete farms[farmer]` statement. This will reset the entire `Farm` struct to its default values.

- The modified function should look like this:

```
function resetFarm(address farmer) private {
    delete farms[farmer];
}
```

Remediation (Revised commit: 974a006): The `delete` the keyword was implemented in the `resetFarm` function.

F-2023-0308 - Cache State Variables - Info

Description:

Cache state variables issues in Solidity refer to situations where developers fail to efficiently manage and update state variables in smart contracts. These issues can lead to suboptimal gas usage, decreased contract performance, and even vulnerabilities that can be exploited by malicious actors. Properly handling and caching state variables is crucial for maintaining efficient and secure smart contracts.

If same state variable is being used in same function as getter more than one time that variable can be cached to save Gas.

```
        if (farm.boosted) {
            amountToRefund = farm.lockedAmount[address(_platformToken)];
            farm.lockedAmount[address(_platformToken)] = 0;

            _platformToken.safeTransfer(farmer, amountToRefund);
        }

        emit FarmEnded(
            farmer,
            farm.boosted,
            farm.startTime,
            farm.lockTime,
            farm.yield
        );
```

For example `farm.boosted` here is used more than one time so it ban be cached.

Assets:

- contracts/Farm.sol [https://github.com/Genieswap-com/simplified-farms-contracts]

Status: Fixed

Recommendations

Recommendation:

Enhance contract efficiency and security by caching state variables in memory when used multiple times in a function. This approach reduces gas consumption and potential vulnerabilities from frequent state variable updates.

Remediation (Revised commit: e67495c): Client cached frequently used state variables to stack variables.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/Genieswap-com/simplified-farms-contracts
Commit	926a0dac1e9dea07e886cb891d5a43f9d4213fb4
Whitepaper	Not provided
Requirements	https://github.com/Genieswap-com/simplified-farms-contracts/docs
Technical Requirements	https://github.com/Genieswap-com/simplified-farms-contracts/docs

Contracts in Scope

./contracts/Farm.sol

./contracts/FarmFactory.sol