MOTECH AUDIT

SMART CONTRACT SECURITY AUDIT

SECURITY ASSESSMENT

2021

NAC PERIOD STAKING FARM AUDIT REPORT

15 NOV 2021

SECURITY ASSESMENT

SUMMARY

This report has been prepared for NAC Period Staking Farm smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognise ed library. A comprehensive examination has been performed, utilising Static Analysis and ManualReview techniques.

The auditing process pays special attention to the following considerations:

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

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

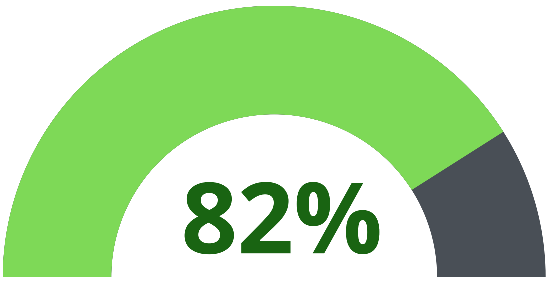
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more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

SECURITY ASSESMENT

SUMMARY

Pass/Fail	Analyzer	Description
Pass	Motech Audit	Vulnerability analyzer built by Motech Audit that quickly searches within smart contracts for vulnerable code fragments not only exact but also syntactically different clones.
Compile Error	Formal Verifier	Sound formal verifier built by Motech Audit that analyzes within smart contracts for proving that a program satisfies nonexistence of vulnerability such as integer overflow.
Failed	Achilles	Symbolic analyzer built by Motech Audit that extracts code patterns within smart contracts for finding syntactic and semantic bugs and vulnerabilities based on SWC specification.

Pass   Compile Error   Failed



File : PeriodStakingFarm.sol

Language : solidity

Size. : 19398 bytes

Date : 2021-11-15T08:41:45.274Z

(STATEMENT : MOTECH AUDIT ONLY ISSUES THIS REPORT BASED ON THE FACT THAT HAS OCCURRED OR EXISTED BEFORE THE REPORT IS ISSUED, AND BEARS THE CORRESPONDING RESPONSIBILITY IN THIS REGARD. FOR THE FACTS OCCUR OR EXIST LATER AFTER THE REPORT, MOTECH AUDIT CANNOT JUDGE THE SECURITY STATUS OF ITS SMART CONTRACT. MOTECH AUDIT IS NOT RESPONSIBLE FOR IT. THE SECURITY AUDIT ANALYSIS AND OTHER CONTENTS OF THIS REPORT ARE BASED ON THE DOCUMENTS AND MATERIALS PROVIDED BY THE INFORMATION PROVIDER TO MOTECH AUDIT AS OF THE DATE OF THIS REPORT (REFERRED TO AS "THE PROVIDED INFORMATION"). MOTECH AUDIT ASSUMES THAT: THERE HAS BEEN NO INFORMATION MISSING, TAMPERED, DELETED, OR CONCEALED. IF THE INFORMATION PROVIDED HAS BEEN MISSED, MODIFIED, DELETED, CONCEALED OR REFLECTED AND IS INCONSISTENT WITH THE ACTUAL SITUATION, MOTECH AUDIT WILL NOT BEAR ANY RESPONSIBILITY FOR THE RESULTING LOSS AND ADVERSE EFFECTS. MOTECH AUDIT WILL NOT BEAR ANY RESPONSIBILITY FOR THE BACKGROUND OR OTHER CIRCUMSTANCES OF THE PROJECT.)

PASSED

SECURITY ASSESMENT

DISCLAIMER

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and MotechAudit and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (MotechAudit) owe no duty of care towards you or any other person, nor does MotechAudit make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and MotechAudit hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, MotechAudit hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against MotechAudit, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

SECURITY ASSESMENT

BACKGROUND

Motech Audit was commissioned by Ark Community to perform an audit of smart contracts:

PeriodStakingFarm.sol
report_7ac3dd24380dce05fa58abe4ec643823bc6f55e129ffc60c5f96e6cee43a81b0

The purpose of the audit was to achieve the following:

- Ensure that the smart contract functions as intended.
- Identify potential security issues with the smart contract.

The information in this report should be used to understand the risk exposure of the smart contract, and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

SECURITY ASSESMENT

VULNERABILITY & RISK LEVEL

29 Total Issues

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Total Issues	Vulnerability	Risk (Required Action)
Critical	0	A vulnerability that can disrupt the contract functioning in a number of scenarios,or creates a risk that the contract may be broken	Immediate action to reduce risk level.
High	3	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	1	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	6	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Note	19	A vulnerability that have informational character but is not affecting any of the code.	An observation that does not determine a level of risk.

SECURITY ASSESMENT

AUDITING STRATEGY AND TECHNIQUES APPLIED

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

Methodology

The auditing process follows a routine series of steps:
1. Code review that includes the following:
- Review of the specifications, sources, and instructions provided to Motech Audit to make sure we understand the size, scope, and functionality of the smart contract.
- Manual review of code, which is the process of reading source code line-byline in an attempt to identify potential vulnerabilities.
- Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Motech Audit describe.

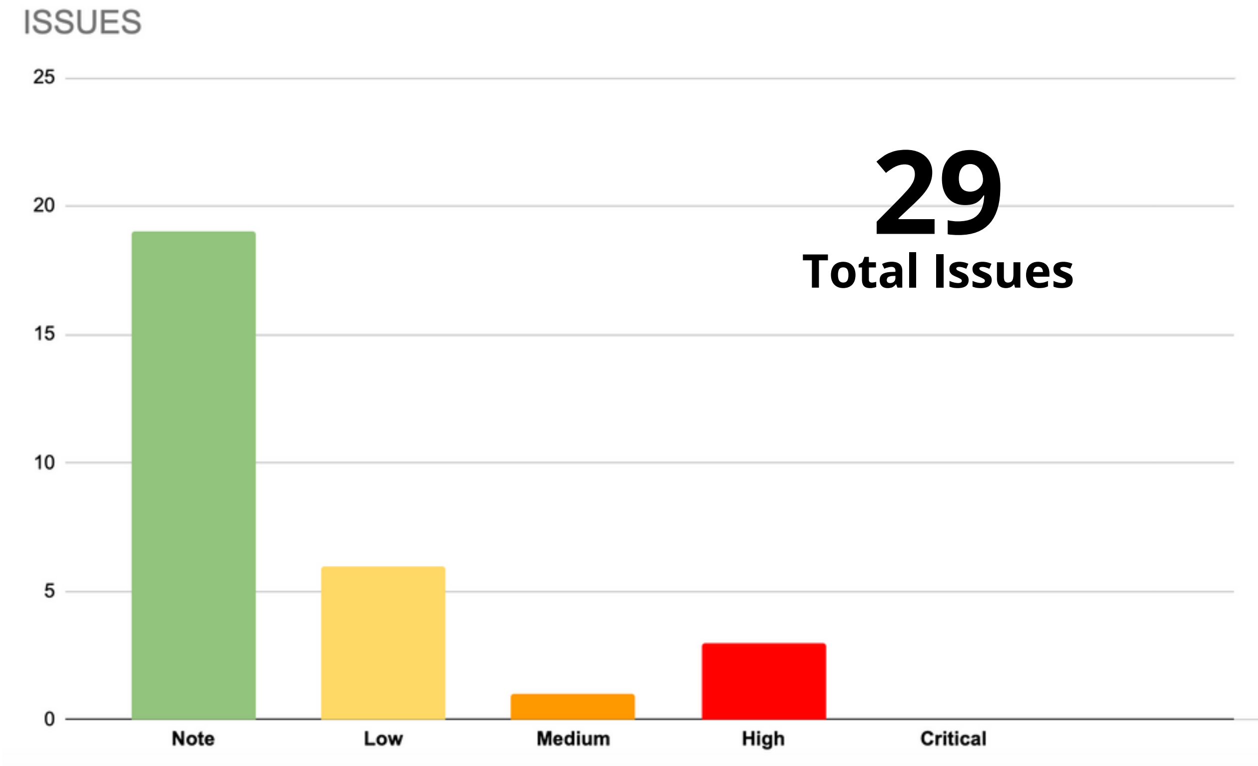
2. Testing and automated analysis that includes the following:
- Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
- Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.

3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

SECURITY ASSESMENT

FINDINGS



Severity	Issue	Code Lines
High	SWC-104	150, 156, 162
Medium	SWC-102	0
Low	SWC-103	3, 21, 93, 144, 169, 219
Note	SWC-108	246, 247, 248, 249, 251
Note	SWC-116	346, 348, 351, 354, 361, 364, 377, 381, 384, 4..

SECURITY ASSESMENT

CODE

1. SWC-104 / lines: 150 High Achilles

A security vulnerability has been detected.

```
149         // bytes4(keccak256(bytes('approve(address,uint256)')));
150         (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value));
151         return (success && (data.length == 0 || abi.decode(data, (bool))));
```

In detail

The return value of a message call is not checked. Execution will resume even if the called contract throws an exception. If the call fails accidentally or an attacker forces the call to fail, this may cause unexpected behaviour in the subsequent program logic.

2. SWC-104 / lines: 156 High Achilles

A security vulnerability has been detected.

```
155         // bytes4(keccak256(bytes('transfer(address,uint256)')));
156         (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value));
157         return (success && (data.length == 0 || abi.decode(data, (bool))));
```

In detail

The return value of a message call is not checked. Execution will resume even if the called contract throws an exception. If the call fails accidentally or an attacker forces the call to fail, this may cause unexpected behaviour in the subsequent program logic.

3. SWC-104 / lines: 162 High Achilles

A security vulnerability has been detected.

```
161         // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
162         (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to,
    value));
163         return (success && (data.length == 0 || abi.decode(data, (bool))));
```

In detail

The return value of a message call is not checked. Execution will resume even if the called contract throws an exception. If the call fails accidentally or an attacker forces the call to fail, this may cause unexpected behaviour in the subsequent program logic.

SECURITY ASSESMENT

CODE

4. SWC-102 / lines: 0 Medium Achilles

A security vulnerability has been detected.

```
// File: contracts/ITRC20.sol
```

In detail

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.

5. SWC-103 / lines: 3 Low Achilles

A security vulnerability has been detected.

```

pragma solidity ^0.5.8;

```

In detail

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

6. SWC-103 / lines: 21 Low Achilles

A security vulnerability has been detected.

```

pragma solidity ^0.5.0;
// pragma experimental ABIEncoderV2;
```

In detail

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

7. SWC-103 / lines: 93 Low Achilles

A security vulnerability has been detected.

```
pragma solidity ^0.5.8;
/**
```

In detail

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

8. SWC-103 / lines: 144 Low Achilles

A security vulnerability has been detected.

```
pragma solidity ^0.5.15;
```

In detail

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

9. SWC-103 / lines: 169 Low Achilles

A security vulnerability has been detected.

```
pragma solidity ^0.5.0;
```

In detail

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

SECURITY ASSESMENT

CODE

10. SWC-103 / lines: 219 Low Achilles

A security vulnerability has been detected.

```
pragma solidity ^0.5.8;
```

In detail

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

11. SWC-108 / lines: 246 Note Achilles

A security vulnerability has been detected.

```
    ITRC20 token = ITRC20(address(0x415BB46A2DA6957D5FE239713A893E81BF24F2817E));
    ITRC20 y = ITRC20(address(0x41C5CFE57926CDBFC8C01EF423FB3A0DD871180CF8));
```

In detail

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

12. SWC-108 / lines: 247 Note Achilles

A security vulnerability has been detected.

```
    ITRC20 token = ITRC20(address(0x415BB46A2DA6957D5FE239713A893E81BF24F2817E));
    ITRC20 y = ITRC20(address(0x41C5CFE57926CDBFC8C01EF423FB3A0DD871180CF8));
    uint256 DAY_SECONDS = 86400;
```

In detail

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

13. SWC-108 / lines: 248 Note Achilles

A security vulnerability has been detected.

```
247     ITRC20 y = ITRC20(address(0x41C5CFE57926CDBFC8C01EF423FB3A0DD871180CF8));
248     uint256 DAY_SECONDS = 86400;
249     uint256 YEAR_SECONDS = 31536000;
```

In detail

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

14. SWC-108 / lines: 249 Note Achilles

A security vulnerability has been detected.

```
248     uint256 DAY_SECONDS = 86400;
249     uint256 YEAR_SECONDS = 31536000;
250     uint256 public totalStakingLP;
```

In detail

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

15. SWC-108 / lines: 251 Note Achilles

A security vulnerability has been detected.

```
250     uint256 public totalStakingLP;
251     uint[7] _products = [7, 15, 30, 90, 180, 360, 720];
252     mapping(uint => bool) private products;
```

In detail

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

16. SWC-116 / lines: 346 Note Achilles

A security vulnerability has been detected.

```
345         if (assetLP.isLive) {
346             assetLP.amountReward =
assetLP.amountReward.add(assetLP.amountNAC.mul(assetLP.rewardRate).mul(block.timestamp.sub(assetLP.lastFarmingT
347             uint256 remainingAmount;
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Tron Chain is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

17. SWC-116 / lines: 348 Note Achilles

A security vulnerability has been detected.

```
347             uint256 remainingAmount;
348             if (assetLP.canRemoveTime < block.timestamp) {
349                 remainingAmount = amountLP;
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Tron Chain is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

CODE

18. SWC-116 / lines: 351 Note Achilles

A security vulnerability has been detected.

```
            } else {
                remainingAmount =
assetLP.amountLP.mul(assetLP.canRemoveTime.sub(block.timestamp)).div(day).div(DAY_SECONDS);
                remainingAmount = remainingAmount.add(amountLP);
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Tron Chain is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

19. SWC-116 / lines: 354 Note Achilles

A security vulnerability has been detected.

```
            }
            assetLP.canRemoveTime =
block.timestamp.add(remainingAmount.mul(DAY_SECONDS.mul(day)).div(assetLP.amountLP.add(amountLP))).add(DAY_SEC
            assetLP.amountNAC = assetLP.amountNAC.add(amountNAC);
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Tron Chain is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

20. SWC-116 / lines: 361 Note Achilles

A security vulnerability has been detected.

```
360                 assetLP.amountReward = 0;
361                 assetLP.canRemoveTime = DAY_SECONDS.mul(day).add(block.timestamp);
362                 assetLP.isLive = true;
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Tron Chain is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

21. SWC-116 / lines: 364 Note Achilles

A security vulnerability has been detected.

```
363             }
364             assetLP.lastFarmingTime = block.timestamp;
365             assetLP.rewardRate = rewardRates[day];
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Tron Chain is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

22. SWC-116 / lines: 377 Note Achilles

A security vulnerability has been detected.

```
        if (isRevoked == false) {
            require(block.timestamp >= assetLP.canRemoveTime, "Can not remove staking at this time");
        }
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Tron Chain is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

23. SWC-116 / lines: 381 Note Achilles

A security vulnerability has been detected.

```
        uint256 amountNAC = amountLP.mul(assetLP.amountNAC).div(assetLP.amountLP);
        uint256 _seconds = block.timestamp.sub(assetLP.lastFarmingTime);
        uint256 earnedTokens = amountNAC.mul(assetLP.rewardRate).mul(_seconds).div(YEAR_SECONDS).div(100);
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Tron Chain is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

24. SWC-116 / lines: 384 Note Achilles

A security vulnerability has been detected.

```
        assetLP.amountReward = assetLP.amountReward.add(earnedTokens);
        assetLP.lastFarmingTime = block.timestamp;
        assetLP.amountLP = assetLP.amountLP.sub(amountLP);
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Tron Chain is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

25. SWC-116 / lines: 451 Note Achilles

A security vulnerability has been detected.

```
        uint _currentTimestamp = currentTimestamp;
        if (currentTimestamp == 0) _currentTimestamp = block.timestamp;
        uint256 _seconds = _currentTimestamp.sub(assetLP.lastFarmingTime);
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Tron Chain is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

CODE

26. SWC-116 / lines: 467 Note Achilles

A security vulnerability has been detected.

```
466                     AssetLP memory assetLP = assetLPs[msg.sender][_products[i]];
467                     uint256 _seconds = block.timestamp.sub(assetLP.lastFarmingTime);
468                     uint256 earnedTokens =
assetLP.amountNAC.mul(assetLP.rewardRate).mul(_seconds).div(YEAR_SECONDS).div(100);
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Tron Chain is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

27. SWC-116 / lines: 470 Note Achilles

A security vulnerability has been detected.

```
469                     sumAmount = sumAmount.add(assetLP.amountReward).add(earnedTokens);
470                     assetLP.lastFarmingTime = block.timestamp;
471                     assetLP.amountReward = 0;
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Tron Chain is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

28. SWC-116 / lines: 487 Note Achilles

A security vulnerability has been detected.

```
        AssetLP memory assetLP = assetLPs[msg.sender][day];
        uint256 _seconds = block.timestamp.sub(assetLP.lastFarmingTime);
        uint256 earnedTokens =
assetLP.amountNAC.mul(assetLP.rewardRate).mul(_seconds).div(YEAR_SECONDS).div(100);
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Tron Chain is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

29. SWC-116 / lines: 490 Note Achilles

A security vulnerability has been detected.

```
        uint256 sumAmount = assetLP.amountReward.add(earnedTokens);
        assetLP.lastFarmingTime = block.timestamp;
        assetLP.amountReward = 0;
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Tron Chain is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

SECURITY ASSESMENT

CONCLUSION

File : PeriodStakingFarm.sol
Language : solidity
Size. : 19398 bytes
Date : 2021-11-15T08:41:45.274Z

Motech Audit note: Please check the disclaimer above and note, the audit makes no statements or warranties on business model, investment attractiveness or code sustainability. The report is provided for the only contract mentioned in the report and does not include any other potential contracts deployed by Owner.

PASSED