CERTIK

Security Assessment

# NEXTYPE.FINANCE

Apr 19th, 2021

## Summary

This report has been prepared for NEXTYPE.FINANCE 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 recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

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

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

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

# Overview

## Project Summary

| | |
|---|---|
| Project Name | NEXTYPE.FINANCE |
| Description | NEXTYPE is an integrated application ecosystem between gaming, NFT and DeFi that can be cross-chain(cross platform). |
| Platform | Heco |
| Language | Solidity |
| Codebase | https://github.com/nextypefinance/miningtycoon/tree/master |
| Commits | 1fe3bb4e88f956e172789c4f73a925a0d395bc6e |

## Audit Summary

| | |
|---|---|
| Delivery Date | Apr 19, 2021 |
| Audit Methodology | Static Analysis, Manual Review |
| Key Components | |

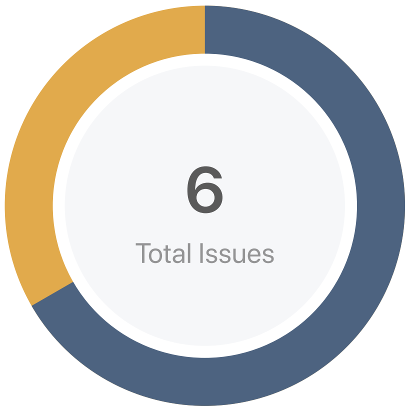
## Vulnerability Summary

| | |
|---|---|
| Total Issues | 6 |
| ● Critical | 0 |
| ● Major | 0 |
| ● Minor | 2 |
| ● Informational | 4 |
| ● Discussion | 0 |

## Audit Scope

| ID | file | SHA256 Checksum |
|---|---|---|
| NXT | NXTP.sol | 1da877d953a747314be8546f7cd7262d28d303f9d2a5f8033c5e66e7d063610c |
| NXP | NXTPFarm.sol | 6d1bc38fc18fadeb151043e864760be171e90db3a343f238eadb2de33f868c13 |

## Findings



| | | |
|---|---|---|
| 🟥 | Critical | **0** (0.00%) |
| 🟧 | Major | **0** (0.00%) |
| 🟨 | Minor | **2** (33.33%) |
| 🟦 | Informational | **4** (66.67%) |
| 🟩 | Discussion | **0** (0.00%) |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| NXP-01 | Missing Check Duplicated Token | Logical Issue | ● Minor | ✓ Resolved |
| NXP-02 | Potential Reentrancy Risk | Volatile Code | ● Minor | ✓ Resolved |
| NXP-03 | Potential Arithmetic Operations Overflow | Volatile Code | ● Informational | ✓ Resolved |
| NXP-04 | Missing Emit Events | Optimization | ● Informational | ✓ Resolved |
| NXP-05 | Lack of Input Validation | Volatile Code | ● Informational | ✓ Resolved |
| NXT-01 | Redundant Codes | Logical Issue | ● Informational | ✓ Resolved |

## NXP-01 | Missing Check Duplicated Token

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Minor | NXTPFarm.sol: 19~21 | ✓ Resolved |

### Description

There is no validation to avoid adding duplicated tokens in function `addAllowedTokens`.

### Recommendation

Consider checking the new token whether added before.

### Alleviation

The development heeded our advice and resolved this issue in commit df6a7fcca53403494e7c523e039d4dbc82b4da37.

## NXP-02 | Potential Reentrancy Risk

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Minor | NXTPFarm.sol: 24~34, 37~43, 66~76, 80~93 | ✓ Resolved |

### Description

There is a potential reentrancy risk on key actions since the implementation of staking tokens are unknown. Examples:

```
24  stakeTokens()
```

```
37  unstakeTokens()
```

```
66  stakeTokens2()
```

```
80  unstakeTokens2()
```

### Recommendation

Consider applying the modifier `nonReentrant` of contact `ReentrancyGuard` of openzeppelin to the above key functions. Example:

```
import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";

function stakeTokens(uint256 _amount, address token, uint256 pool) public nonReentrant {
    ...
}
```

### Alleviation

The development heeded our advice and resolved this issue in commit df6a7fcca53403494e7c523e039d4dbc82b4da37.

## NXP-03 | Potential Arithmetic Operations Overflow

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Informational | NXTPFarm.sol: 31~32, 73~74 | Resolved |

### Description

Arithmetic operation by plain arithmetic operators may cause overflow.

### Recommendation

Consider using functions of `SafeMath` to safeguard the arithmetic operations.

```
stakingBalance[pool][token][msg.sender] =
  stakingBalance[pool][token][msg.sender].add(_amount);
```

```
stakingBalance2[pool][token1][msg.sender] = stakingBalance2[pool][token1]
[msg.sender].add(_amount1);
stakingBalance2[pool][token2][msg.sender] = stakingBalance2[pool][token2]
[msg.sender].add(_amount2);
```

### Alleviation

The development heeded our advice and resolved this issue in commit df6a7fcca53403494e7c523e039d4dbc82b4da37.

## NXP-04 | Missing Emit Events

| Category | Severity | Location | Status |
|---|---|---|---|
| Optimization | ● Informational | NXTPFarm.sol: 19~21, 24~34, 37~43, 66~76, 80~93 | Resolved |

### Description

Several key actions are defined without event declarations. Examples:

```
addAllowedTokens();
```

```
stakeTokens();
```

```
unstakeTokens();
```

```
stakeTokens2();
```

```
unstakeTokens2();
```

### Recommendation

Consider emitting events for key actions. Example:

```
event AddAllowedTokens(address token);
function addAllowedTokens(address token) public onlyOwner {
  allowedTokens.push(token);
  emit AddAllowedTokens(token);
}
```

### Alleviation

The development heeded our advice and resolved this issue in commit df6a7fcca53403494e7c523e039d4dbc82b4da37.

## NXP-05 | Lack of Input Validation

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Informational | NXTPFarm.sol: 19~21 | ✓ Resolved |

### Description

The passed parameter `token` should be verified as a non-zero value to prevent being mistakenly assigned as `address(0)` in functions `addAllowedTokens` and `unstakeTokens`.

The passed parameter `token1` and `token2` should be verified as a non-zero value to prevent being mistakenly assigned as `address(0)` in functions `unstakeTokens2`.

### Recommendation

Check that the address is not zero by adding following check.

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

### Alleviation

The development heeded our advice and resolved this issue in commit df6a7fcca53403494e7c523e039d4dbc82b4da37.

## NXT-01 | Redundant Codes

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Informational | NXTP.sol: 21 | ✓ Resolved |

### Description

The function call `transferOwnership(_msgSender());` is redundant since it is already done in the constructor of parent contract `Ownable`:

```
//Ownable.sol
constructor () internal {
    address msgSender = _msgSender();
    _owner = msgSender;
    emit OwnershipTransferred(address(0), msgSender);
}
```

### Recommendation

Consider removing the function call `transferOwnership(_msgSender());`.

### Alleviation

The development heeded our advice and resolved this issue in commit df6a7fcca53403494e7c523e039d4dbc82b4da37.

# Appendix

## Finding Categories

### Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

### Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

### Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

### Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

### Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

### Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in storage one.

### Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete .

### Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

## Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

## Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

## Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

## Disclaimer

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

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

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

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

## About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.