CERTIK

Security Assessment

Faraland

May 23rd, 2021

Table of Contents

Summary

Overview

Project Summary

Audit Summary

Vulnerability Summary

Audit Scope

Findings

ECK-01 : Incorrect Fee

ECK-02 : Potential Over Mint

ECK-03 : Corner Case for Non-Contract Caller Check

ECK-04 : Centralization Risk

ECK-05 : Lack of Upper Bound Check for Input Variable

MKC-01 : Redundant Comparison to Boolean Constant

MKC-02 : Lack of Reentrancy Check

MKC-03 : Lack of Upper Bound Check for Input Variable

Formal Verification Requests

Appendix

Disclaimer

About

Summary

This report has been prepared for Faraland 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 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	Faraland
Platform	BSC
Language	Solidity
Codebase	https://github.com/ensete/moonknights-sc
Commits	1. d4f0e82b0dac3e32d26d05c124c5f772647bb321 2. f0e0531b1989d1e20df7526ac824931b23215dd5

Audit Summary

Delivery Date	May 23, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Total Issues	8
● Critical	0
● Major	1
● Medium	3
● Minor	2
● Informational	2
● Discussion	0

Audit Scope

ID	file	SHA256 Checksum
ECK	equipment/Equipment.sol	1f177043682bd4e6eec0aad75134486c23b2942f83f4e8c1124d686043c22900
ERC	equipment/libs/ERC1155.sol	e6856fd10e60796174091a447998c822e996bd23dcd9e7e0997be1d832fbd234
IEC	interfaces/IEquipment.sol	3a133857aeda8326f758da8838355a8009847a4f56a509485f4ccf4013d90874
IMK	interfaces/IMoonKnight.sol	2a7dfa5e5a9312d8f624088b2b96318d52037ad84da75169b589d2f444e584c0
IPC	interfaces/IPet.sol	a393e0250039d4b11c21b9237ec8a99abde4be4bbf9b4c2df8bccf919e573881
ISE	interfaces/IStakingExpPool.sol	6946f59a5a48fa82fc2cec083a91e9d94894412fdcf2d0c6a9b8a158fe92679e
MKC	knight/MoonKnight.sol	fc12f0dddeec2f7c769522c999594e13731d4ab5dfd271703f5522ca10e96706
SEP	staking/StakingExpPool.sol	fe9251130eeb624a7889b87ed3bd309302ec6001b6bf8c1dbae12c394498ebcf
FCC	token/FaraCrystal.sol	964f12fce860c551eca1c5e0641222bb79ec36fb667e75da0a77cad50babb77f
ATC	utils/AcceptedToken.sol	b7d3c2591193424c20e9504a73aa93fdfa934919b69b02c75b815cf8b3c91aa9
PGC	utils/PermissionGroup.sol	09eb3d5ff25cfeea336541cdc93c3142c6fedbf89f8cb8bddb61287e7b9edcb5
TWC	utils/TokenWithdrawable.sol	6213143a6ec830cd18f8b9517412d1bcf467d0a58d57408e6f654dcf64a4e2c1

To set up the project correctly, improve overall project quality and preserve upgradability, the following roles, are adopted in the codebase:

`operator`, is adopted to mint, return, and equip equipment in `Equipment`.

`operator`, is adopted to generate and level up knights in `MoonKnight`.

`owner`, is adopted to update configurations, fees and create items in `Equipment`.

`owner`, is adopted to set up and manage the vendor in `EquipmentVendor`.

`owner`, is adopted to update configurations and fees in `MoonKnight`.

`owner`, is adopted to set MoonKnight contract in `StakingExpPool`.

`owner`, is adopted to set the accepted token in `AcceptedToken`.

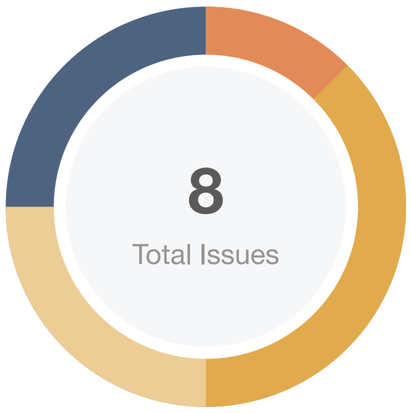
`owner`, is adopted to manage operators in `PermissionGroup`.

`owner`, is adopted to manage the blacklist in `TokenWithdrawable`.

`knightOwner`, is adopted to manage users' own knights in `MoonKnight`.

To improve the trustworthiness of the project, any dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of `Timelock` contract.

Findings



🟥	**Critical**	**0** (0.00%)
🟧	**Major**	**1** (12.50%)
🟨	**Medium**	**3** (37.50%)
🟨	**Minor**	**2** (25.00%)
🟦	**Informational**	**2** (25.00%)
🟩	**Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
ECK-01	Incorrect Fee	Logical Issue	● Medium	✓ Resolved
ECK-02	Potential Over Mint	Logical Issue	● Medium	✓ Resolved
ECK-03	Corner Case for Non-Contract Caller Check	Volatile Code	● Informational	ⓘ Acknowledged
ECK-04	Centralization Risk	**Centralization / Privilege**	● **Medium**	ⓘ **Acknowledged**
ECK-05	Lack of Upper Bound Check for Input Variable	Logical Issue	● Minor	ⓘ Acknowledged
MKC-01	Redundant Comparison to Boolean Constant	Coding Style	● Informational	✓ Resolved
MKC-02	Lack of Reentrancy Check	Logical Issue	● Major	✓ Resolved
MKC-03	Lack of Upper Bound Check for Input Variable	Logical Issue	● Minor	ⓘ Acknowledged

ECK-01 | Incorrect Fee

Category	Severity	Location	Status
Logical Issue	● Medium	equipment/Equipment.sol: 122	✓ Resolved

Description

`rollEquipmentGacha()` will mint random equipment. However it charges `upgradeFeeInToken`, rather than `mintFeeInToken`.

Recommendation

We advise the client to double check the fee mechanism to make sure the fee charging is reasonable.

Alleviation

The client heeded the advice and resolved this issue in commit `f0e0531b1989d1e20df7526ac824931b23215dd5`.

ECK-02 | Potential Over Mint

Category	Severity	Location	Status
Logical Issue	● Medium	equipment/Equipment.sol: 109~110	✓ Resolved

Description

It is dangerous to change `balances` without checking `item.maxSupply` and `item.minted`. Suppose an item has the property `maxSupply=10` and `minted=9`. The operator then call `mint(account, id, 1)`. This will update `item.minted=10`. Users can still call `upgradeItem()` to burn the last tier of this item to mint this item. This will cause `minted > maxSupply`.

Recommendation

We advise the client to check `item.maxSupply` and `item.minted` before updating `balances` and `_items.minted`.

Alleviation

The client heeded the advice and resolved this issue in commit `f0e0531b1989d1e20df7526ac824931b23215dd5`.

ECK-03 | Corner Case for Non-Contract Caller Check

Category	Severity	Location	Status
Volatile Code	● Informational	equipment/Equipment.sol: 117	ⓘ Acknowledged

Description

`isContract()` cannot 100% guarantee the caller is a non-contract user, since `EXTCODESIZE` returns 0 if it is called from the constructor of another contract. Please consider if this is a problem for the project.

Recommendation

We advise the client to be skeptical about the return value of `isContract()`.

Alleviation

N/A

ECK-04 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Medium	equipment/Equipment.sol: 125, 145, 138	ⓘ Acknowledged

Description

The role `operator` has the authority to mint and burn arbitrary equipment. In `returnItems()`, the operator can even mint equipment without the limitation of `item.maxSupply`.

Recommendation

We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions:

1. `Timelock` with reasonable latency for community awareness on privileged operations;
2. Multisig with community-voted 3rd-party independent co-signers;
3. DAO or Governance module increasing transparency and community involvement;

Alleviation

[**Faraland Team**]: After the system is running smoothly, we will consider to pass the owner key to our community through a DAO contract, that can adjust all the parameters in the SC. Operators are trusted to operate for the best interest of the whole platform.

ECK-05 | Lack of Upper Bound Check for Input Variable

Category	Severity	Location	Status
Logical Issue	● Minor	equipment/Equipment.sol: 48, 53	ⓘ Acknowledged

Description

The role `owner` can set the following state variables arbitrary large causing potential risks in fees :

- `upgradeFeeInToken`
- `mintFeeInToken`

Recommendation

We recommend setting upper bound and check the input variable `fee`.

Alleviation

[**Faraland Team**]: We want these parameters to be flexible based on the price of our token.

MKC-01 | Redundant Comparison to Boolean Constant

Category	Severity	Location	Status
Coding Style	● Informational	knight/MoonKnight.sol: 145~146, 165	✓ Resolved

Description

`_validateStr()` returns a Boolean value. Boolean value can be used directly and do not need to be compare to true or false. For example, `require(_validateStr("certik",false))` is valid.

Recommendation

We recommend removing the equality to the Boolean constant.

Alleviation

The client heeded the advice and resolved this issue in commit `f0e0531b1989d1e20df7526ac824931b23215dd5` .

MKC-02 | Lack of Reentrancy Check

Category	Severity	Location	Status
Logical Issue	● Major	knight/MoonKnight.sol: 212, 280, 239, 265	✓ Resolved

Description

Calling `MoonKnight.buy()`, `MoonKnight.sacrificeKnight()`, `MoonKnight.cancelOffer()` and `MoonKnight.takeOffer()` might trigger function `address.call{}()` , which is implemented by the third party. If there are vulnerable external calls in the third party, reentrancy attacks could be conducted because these four functions have state updates and event emits after external calls.

The scope of the audit would treat the third-party implementation as a black box and assume its functional correctness. However, third parties may be compromised in the real world that leads to assets lost or stolen.

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - nonReentrant modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

The client heeded the advice and resolved this issue in commit `f0e0531b1989d1e20df7526ac824931b23215dd5` .

MKC-03 | Lack of Upper Bound Check for Input Variable

Category	Severity	Location	Status
Logical Issue	● Minor	knight/MoonKnight.sol: 106, 110	ⓘ Acknowledged

Description

The role `owner` can set the following state variables arbitrary large or small causing potential risks:

- `setFloorPriceCap`
- `setServiceFee`

Recommendation

We recommend setting proper ranges and check the input variable `value`.

Alleviation

[**Faraland Team**]: We want these parameters to be flexible based on the price of our token.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Logical Issue

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

Volatile Code

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

Coding Style

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

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

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.

CERTIK