Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Fundora

Veridise Inc.

May 24, 2024

► **Prepared For:**

Ginger Joy
https://gingerjoy.gs

► **Prepared By:**

Nicholas Brown
Bryan Tan

► **Contact Us:** contact@veridise.com

► **Version History:**

May. 24, 2024 V1.01 - Updated issue statuses
May. 03, 2024 V1

Contents

1 Executive Summary

From Apr. 29, 2024 to May. 1, 2024, Ginger Joy engaged Veridise to review the security of their Fundora smart contracts. The review covered a series of ERC 20 and ERC 721 tokens that represent ownership of digital assets used with the Funodra video game. Veridise conducted the assessment over 6 person-days, with 2 engineers reviewing code over 3 days on commit `f2317ff`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual code review.

Code assessment. The Fundora developers provided the source code of the Fundora contracts for review. The source code extends the OpenZeppelin implementations of ERC20 and ERC721 with some minor modifications and project-specific features. It contains some documentation in the form of READMEs and documentation comments on functions and storage variables.

The source code contained a test suite, which the Veridise auditors noted covers most the extended token functionality in both successful and unsuccessful scenarios.

Summary of issues detected. The audit uncovered 5 issues, comprising 1 low-severity issue, 2 warnings, and 2 informational issues. This includes a centralization risk with respect to owning the tokens (V-GJF-VUL-001) as well as several maintainability concerns that could lead to future developer errors (V-GJF-VUL-003, V-GJF-VUL-005). Ginger Joy has resolved all of the identified issues.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Fundora	`f2317ff`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Apr. 29 - May. 1, 2024	Manual & Tools	2	6 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	0	0	0
Low-Severity Issues	1	1	0
Warning-Severity Issues	2	2	2
Informational-Severity Issues	2	2	2
TOTAL	5	5	4

Table 2.4: Category Breakdown.

Name	Number
Maintainability	4
Access Control	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Fundora's smart contracts. In our audit, we sought to answer questions such as:

- Is it possible for a malicious user to steal tokens from another user?
- Is it possible for a malicious user to prevent other users from using/transferring their tokens?
- Is it possible for a single compromised private key to allow an attacker to take over the protocol?
- Are there any maintainability issues that could lead to future errors?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard, as well as the open-source tool Slither. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.

Scope. The scope of this audit is limited to the `contracts` folder of the source code provided by the Fundora developers, which contains the smart contract implementation of the Fundora smart contracts.

Methodology. Veridise auditors inspected the provided tests and read the Fundora documentation. They then began a manual review of the code assisted by both static analyzers and automated testing.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-GJF-VUL-001	Centralization risks	Low	Acknowledged
V-GJF-VUL-002	Confusing error message	Warning	Fixed
V-GJF-VUL-003	Code for treasury operative check is duplicated...	Warning	Fixed
V-GJF-VUL-004	FundoraOperative can be declared abstract	Info	Fixed
V-GJF-VUL-005	Duplicated logic in some contracts	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-GJF-VUL-001: Centralization risks

Severity	Low	**Commit**	f2317ff
Type	Access Control	**Status**	Acknowledged
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	N/A		

Similar to many projects, the `Fundora`'s token contracts inherit from OpenZeppelin's `OwnableUpgradable` contract, which defines an `owner` role that is given special permissions. In particular, the owner is given the following abilities:

- The owner of the contract can change the implementation of the Fundora contracts by authorizing an upgrade.
- The owner of the contract can assign roles to other accounts.

Furthermore, the concrete Fundora token contracts such as `FundoraBanner`, `FundoraDiamond`, etc. all extend from a custom `FundoraOperative` contract, which defines a `treasury` role that can mint tokens.

These roles introduce centralization risks that could be harmful to users if abused.

Impact If a private key of the `owner` account or the account with the `treasury` role were stolen, a hacker would have access to sensitive functionality that could compromise the project. For example, an attacker with access to the `treasury` account could mint arbitrary amounts of tokens for themselves, and an attacker with access to the `owner` account could change the implementation of the contract.

Recommendation As these are all particularly sensitive operations, we would encourage the developers to utilize a decentralized governance or multi-sig contract as opposed to a single account, which introduces a single point of failure.

Developer Response The developers plan to use a multi-sig wallet for the `owner` when deploying the protocol to testnet and mainnet.

4.1.2 V-GJF-VUL-002: Confusing error message

Severity	Warning	**Commit**	f2317ff
Type	Maintainability	**Status**	Fixed
File(s)	See description		
Location(s)	batchTransfer()		
Confirmed Fix At	ae018e6		

The `FundoraCharSkin.batchTransfer()` function contains a `require` statement that checks that its `tokenIdList` argument is non-empty. If the check fails, the transaction will revert with an error message stating: `"FundoraBanner: Token id list is empty."`. This message indicates that an error occurred with the `FundoraBanner` contract, even though it actually occurred in the `FundoraCharSkin` contract.

```
function batchTransfer(address to, uint256[] calldata tokenIdList) public {
        require(tokenIdList.length > 0, "FundoraBanner: Token id list is empty."); //
    Ensure the token ID list is not empty
```

Snippet 4.1: Snippet from `batchTransfer()` that performs the check

This issue also impacts the following additional locations:

- `FundoraTool.batchTransfer()`
- `FundoraToolSkin.batchTransfer()`

Impact These error messages may be confusing for users and future developers.

Recommendation Indicate the correct contract that raises the error in the error message.

Developer Response The `batchTransfer()` method was moved to the new `FundoraNFT` contract introduced by the fix to V-GJF-VUL-005, and the error message now indicates `FundoraNFT` as the contract.

4.1.3 V-GJF-VUL-003: Code for treasury operative check is duplicated in many places

Severity	Warning	**Commit**	f2317ff
Type	Maintainability	**Status**	Fixed
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	ae018e6		

In many functions in the project, the modifier call `onlyOperative("treasury")` is used to check that the caller is the "treasury operative". The problem is that the "treasury" string is hardcoded into each one of these modifier calls, which is an anti-pattern.

```
1 function safeMint(address to, uint256 tokenId) public onlyOperative("treasury") {
```

Snippet 4.2: Example of the `onlyOperative("treasury")` modifier in the declaration of `safeMint()`.

This modifier call is used in the following locations:

- `FundoraBanner.safeMint` (`contracts/FundoraBanner.sol:40`)
- `FundoraBanner.batchMint` (`contracts/FundoraBanner.sol:49`)
- `FundoraCharSkin.safeMint` (`contracts/FundoraCharSkin.sol:40`)
- `FundoraCharSkin.batchMint` (`contracts/FundoraCharSkin.so:49`)
- `FundoraDiamond.deposit` (`contracts/FundoraDiamond.sol:62`)
- `FundoraGold.deposit` (`contracts/FundoraGold.sol:62`)
- `FundoraTool.safeMint` (`contracts/FundoraTool.sol:41`)
- `FundoraTool.batchMint` (`contracts/FundoraTool.sol:50`)
- `FundoraToolSkin.safeMint` (`contracts/FundoraToolSkin.sol:40`)
- `FundoraTookSkin.batchMint` (`contracts/FundoraToolSkin.so:49`)

Impact Hardcoding the "treasury" string into each `onlyOperative` modifier call is error-prone and can lead to maintenance problems. For example, if additional functions are added that should only be callable by the treasury, then it may be possible for a developer to accidentally introduce a typo such as `onlyOperative("tresaury")` when typing the modifier call. This could potentially lead to denial-of-service problems.

Recommendation Define a modifier like `onlyOperativeTreasury` for commonly used roles. By doing so, the Solidity compiler could catch problems such as the ones mentioned above.

Developer Response The developers defined a constant named `TREASURY` and changed all calls to `onlyOperative("treasury")` to use the constant instead.

4.1.4 V-GJF-VUL-004: FundoraOperative can be declared abstract

Severity	Info	**Commit**	f2317ff
Type	Maintainability	**Status**	Fixed
File(s)	`contracts/FundoraOperative.sol`		
Location(s)	N/A		
Confirmed Fix At	ae018e6		

The `FundoraOperative` contract defines common access control features for the smart contracts in the project. Because `FundoraOperative` does not have any useful functionality by itself, it does not appear to be intended to be deployed on its own. Thus, it should likely be declared as an abstract contract.

```
1 contract FundoraOperative is OwnableUpgradeable {
```

Snippet 4.3: Declaration of `FundoraOperative`

Impact Since the `FundoraOperative` contract is not declared as abstract, it is possible for it to be deployed, even though it does not serve any useful purpose by itself. This could result in a waste of gas.

Recommendation Declare `FundoraOperative` as an abstract contract.

Developer Response The developers applied the recommendation.

4.1.5 V-GJF-VUL-005: Duplicated logic in some contracts

Severity	Info	**Commit**	f2317ff
Type	Maintainability	**Status**	Fixed
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	ae018e6		

The `safeMint()`, `batchMint()`, `batchTransfer()`, `_authorizeUpgrade()`, and `fuse()` functions have duplicated functionality in the following contracts:

- `FundoraBanner`
- `FundoraCharSkin`
- `FundoraToolSkin`

The function definitions could be moved to `FundoraFusable` to avoid code duplication. Note that `FundoraTool` is not `Fusable`, but it contains the same functionality for all the above functions except `fuse()`.

Furthermore, `FundoraGold` and `FundoraDiamond` also have duplicated functionality for the `deposit()` and `withdraw()` functions.

Impact This code duplication is error-prone and can decrease the readability of the code bad. Specifically, if a developer changes one implementation in the future, they will need to remember to also modify the other implementations. Otherwise, it will be easy to accidentally introduce inconsistencies between the different contracts.

Recommendation Combine common functionality into an inherited contract. This could be the existing `FundoraFusable` for some of the above functionality, or new abstract contracts.

Developer Response The developers refactored the token contracts to inherit from two new contracts `FundoraToken` and `FundoraNFT`, which contain the common functionality.

Glossary

ERC 20 The famous Ethereum fungible token standard. See `https://eips.ethereum.org/EIPS/eip-20` to learn more. 1

ERC 721 The Ethereum non-fungible token standard. See `https://eips.ethereum.org/EIPS/eip-721` to learn more. 1