Security Audit Report for LaunchPad Contracts

Date: Feb 27, 2024

Version: 1.0

Contact: contact@blocksec.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2024.02.29 13:54:58 +08'00'

Contents

Report Manifest

Item	Description
Client	Impossible Finance
Target	LaunchPad Contracts

Version History

Version	Date	Description
1.0	Feb 27, 2024	First Version

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The focus of this audit is on the LaunchPad Contracts [1], which are part of a launchpad project by Impossible Finance. Users can participate in the token pre-sale for this project. Note that the audit scope only includes `IFFixedSale` and its dependencies in the repository.

The auditing process is iterative. Specifically, we will audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following. Our audit report is responsible for the only initial version (i.e., `Version 1`), as well as new codes (in the following versions) to fix issues in the audit report.

Project	Version	Commit SHA
LaunchPad Contracts	`Version 1`	`b30be3bfd3278375d68f52fc48d3485988890a8c`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

[1] https://github.com/ImpossibleFinance/launchpad-contracts

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

∗ Reentrancy
∗ DoS
∗ Access control
∗ Data handling and data flow
∗ Exception handling
∗ Untrusted external call and control flow
∗ Initialization consistency
∗ Events operation
∗ Error-prone randomness
∗ Improper use of the proxy system

1.3.2 DeFi Security

∗ Semantic consistency
∗ Functionality consistency
∗ Access control
∗ Business logic
∗ Token operation
∗ Emergency mechanism
∗ Oracle security
∗ Whitelist and blacklist
∗ Economic impact
∗ Batch transfer

1.3.3 NFT Security

∗ Duplicated item
∗ Verification of the token receiver
∗ Off-chain metadata security

1.3.4 Additional Recommendation

∗ Gas optimization

* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In summary, we did not find any potential security issues. However, there are **three** notes that need to be considered.

- Note: 3

ID	Severity	Description	Category	Status
1	-	Potential centralization risks	Note	-
2	-	Supporting weird ERC20s may introduce accounting issues	Note	-
3	-	Third parties must avoid using controllable parameters in `checkWhitelist`	Note	-

The details are provided in the following sections.

2.1 Notes

2.1.1 Potential centralization risks

Introduced by `Version 1`

Description The `IFFixedSale` contract inherits from `IFWhitelistable` to utilize the whitelist-related functions. Specifically, by setting the `whitelistRootHash`, the pre-sale participants can be restricted. However, the `onlyWhitelistSetterOrOwner` modifier permits the `whitelistSetter` or the owner of the contract to modify the `whitelistRootHash` at any time, which introduces centralization risks.

```
function checkWhitelist(address user, bytes32[] calldata merkleProof, uint256 allocation)
    public
    view
    returns (bool)
{
    // compute merkle leaf from input
    bytes32 leaf = keccak256(abi.encodePacked(user, allocation));

    // verify merkle proof
    return MerkleProof.verify(merkleProof, whitelistRootHash, leaf);
}
```

Listing 2.1: IFFixedSale.sol

```
function setWhitelist(bytes32 _whitelistRootHash)
public
onlyWhitelistSetterOrOwner
{
whitelistRootHash = _whitelistRootHash;

emit SetWhitelist(_whitelistRootHash);
}
```

Listing 2.2: IFWhitelistable.sol

2.1.2 Supporting weird ERC20s may introduce accounting issues

Introduced by `Version 1`

Description The protocol should avoid supporting *weird* ERC20 tokens [1] that may introduce potential security risks. For instance, if `paymentToken` is a deflationary token, the actual amount received may differ from the amount recorded in the contract due to the token's deflationary mechanisms.

2.1.3 Third parties must avoid using controllable parameters in `checkWhitelist`

Introduced by `Version 1`

Description The `checkWhitelist` function in `IFFixedSale` has a potential security issue. If the parameters are controllable, a malicious user could manipulate them to bypass the `MerkleProof.verify` function [2]. It should be noted that the project correctly uses the function in all places because the `user` parameter cannot be manipulated. However, we must notify any third parties about this potential risk.

[1] https://github.com/d-xo/weird-erc20

[2] https://www.rareskills.io/post/merkle-tree-second-preimage-attack