verichains

SECURITY AUDIT OF

MAVIS LAUNCHPAD CONTRACTS

Public Report

Jun 3, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Jun 3, 2024. We would like to thank the Ronin for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Mavis Launchpad contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team found no vulnerabilities in the given version of Mavis Launchpad contracts, only some notes and recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Mavis Launchpad contracts

Mavis Market is a marketplace where users can buy, sell, and showcase NFTs powered by the Ronin network. As the marketplace caters to NFTs in the Ronin network, only the Ronin wallet can be used to buy, sell, and trade in the Mavis Market.

Mavis launchpad contracts designed to empower NFT projects, provides the ideal avenue to raise funds by selling NFT items. It facilitates two distinctive launchpad mechanics:

- First-Come-First-Serve: This method features fixed-price sales complemented by whitelist tier support.
- Mystery Box: NFTs are artfully concealed within boxes, with each box containing NFTs of diverse types and pricing. Users purchase these mystery boxes, with their anticipation building as they reveal the hidden treasures within.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Mavis Launchpad contracts.

It was conducted on the source code provided by the Ronin team:

SHA256 Sum	File
8cd747c9594c3c8e736724b8fa6492df513e99f1da8ead3a53127a81f223aa49	launchpad-contracts-src-v0.1.7.zip

The Mavis Launchpad contracts use `INFTLaunchpad` contract to mint NFTs for users which is out of scope in this audit.

The following files were made available in the course of the review:

SHA256 Sum	File
c1f4317e5f3be9ef4e46f8367066370f27f09674	core/LaunchPausable.sol
14b3bfab558de96dc17f78b508b8c6aeb10a06ed	core/CoreLaunchpad.sol
a0dd8afa61b488ced54441d53955e74bf5cd9882	MavisLaunchpad.sol
7ffb370b1ed029abf7da08225eff0ca5a131d629	LaunchpadGetterSetter.sol
bb69c8f61c1fb794c3ff9e4765e8823d873709d8	SwapForwarder.sol

ac405cdfcc4dc16c6899ab9b25bf8e0a5963950b	libraries/LibAllowlistManagement.sol
5b8d16450825fd50a669a1f4cac5b80db5849fda	libraries/LibGeneralConfig.sol
30320ae78e0cfa96050b415eb1b6b50535bcfcf6	libraries/LibCast.sol
b75a0ba6d5523abd3b70e6c82f85752c7e450628	libraries/LibLaunchpad.sol
0d72fedc39c4d3dbc58e196658b789cfc5a94d30	libraries/LibQuantityTracking.sol
bedf1659010a4ab199de10835acd0bc3cf69b568	libraries/LibConstantShared.sol
adc43f6c0c7c5bb2cec2982ecabe076672170cb9	libraries/LibTierSetting.sol
020c5576f6f856f22a168aef0f1c149f37847790	libraries/LibPayment.sol
adab8ca460d2bcb22e5187f05987ba5395a3fc53	libraries/LibOverriddenValue.sol
64e94eaeb8b3a37327527d7b4235497423a6c171	stage-logic/BaseStageLogic.sol
e469dca0d8a117725244526f14a907885dddbcb5	stage-logic/PublicStageLogic.sol
5aaafc57be5ea60e1745c5900fff7226c758f2e2	stage-logic/AllowlistStageLogic.sol
46b69b53c3d48a54e1f8f8377afe1d6693a0610f	stage-logic/TokenGatedStageLogic.sol

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops

- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Ronin acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. Ronin understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Ronin agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the Ronin will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Ronin, the final report will be considered fully accepted by the Ronin without the signature.

2. AUDIT RESULT

2.1. Overview

The Mavis Launchpad contracts was written in `Solidity` language, with the required version to be `^0.8.22`.

2.1.1. MavisLaunchpad contract

This is the main contract that handles the launchpad mechanics. Operators can create a launchpad for each NFT collection. Each launchpad can have multiple stages, each with a different type:

- `PUBLIC`: Open to anyone.
- `ALLOWED_LIST`: Open to a specific list of addresses, with multiple tiers that the operator can update and assign to each user.
- `TOKEN_GATED`: Requires users to hold specific NFTs from a designated collection. The operator sets the collection address and mint amount limit for each NFT.
- `SERVER_SIGNED`: Not yet implemented in the current version.

Each stage type is a separate contract, allowing the operator to use the MavisLaunchpad to add, remove, or update the logic of any stage without affecting the others.

Users participate in each stage by calling the `execute` function, which delegates the call to the relevant stage contract. The contract then mints the NFTs, collects payment, and distributes payment and fees to the corresponding parties.

2.1.2. Stage logic contracts

These contracts implement the logic for each stage type. The `MavisLaunchpad` contract delegates calls to these contracts to handle the logic for each stage, depending on its type.

All stage logic contracts inherit from `BaseStageLogic`, which contains the logic for:

- Checking the mint limit for each user.
- Handling payment logic.
- Handling mint logic.

Each stage logic contract then implements its own logic to handle:

- The price of the current stage.
- Any special requirements for participation.

2.1.2.1. Public stage

This stage is open to anyone. The operator sets the price and mint limit. There are no special requirements, and each launchpad has only one public stage.

2.1.2.2. Allowed List stage

This stage is open to a specific list of addresses. The users need to be in the list to participate in this stage.

There are also many tiers in this type of stage. Each tier has a different price and mint limit. If users belong to a tier, they will be charged the tier's price and have its mint limit; otherwise, default price and mint limit of the stage will be applied.

2.1.2.3. Token Gated stage

This stage requires users to hold specific NFTs from a designated collection. Each NFT held gives the owner a limited number of NFTs they can mint in this stage.

2.1.3. SwapForwarder Contract

This contract is a utility contract that allows users to swap their tokens for another payment token, which they can then use to participate in the launchpad. The contract calls `MavisLaunchpad` to get the price in the current stage, uses `Katana Swap` to swap input tokens for the required tokens, and then calls `MavisLaunchpad` again to mint the NFTs.

2.2. Findings

During the audit process, the audit team found no vulnerabilities in the given version of Mavis Launchpad contracts, only some notes and recommendations.

2.2.1. Should extend ReentrancyGuardUpgradeable for upgradable contracts INFORMATIVE

Affected files:

- CoreLaunchpad.sol
- SwapForwarder.sol

The `CoreLaunchpad` and `SwapForwarder` contracts are upgradable and deployed behind a proxy contract. Because they extend `ReentrancyGuard`, the `_status` variable is set redundantly during the implementation contract's construction. This leads to higher gas costs and doesn't properly initialize the `_status` variable in the proxy's storage.

```
abstract contract ReentrancyGuard {
    ...
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;

    uint256 private _status;

    constructor() {
```

```
        _status = _NOT_ENTERED;
    }
    ...
}
```

RECOMMENDATION

To optimize the contracts and ensure correct `_status` initialization, follow these steps:

- Modify the `CoreLaunchpad` and `SwapForwarder` contracts to extend `ReentrancyGuardUpgradeable` instead of `ReentrancyGuard`.
- Call the `__ReentrancyGuard_init` function from within the proxy contract's initialize function.

The `ReentrancyGuardUpgradeable` contract is specifically designed for use with upgradable contracts and proxies. By calling `__ReentrancyGuard_init` within the proxy's `initialize` function, you ensure that the reentrancy protection status is correctly initialized within the proxy's storage, saving gas and promoting proper contract behavior.

2.2.2. Add `nonReentrant` for `swapTokenForTokenAndExecute` INFORMATIVE

Affected files:

- SwapForwarder.sol

Although the current contracts are not vulnerable to reentrancy attacks, it is good practice to add a `nonReentrant` modifier to the `swapTokenForTokenAndExecute` function. This modifier will prevent reentrancy attacks in the future development and provide additional security for the contract.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jun 3, 2024*	Public Report	Verichains Lab

Table 2. Report versions history