SECURITY AUDIT OF

GOMU SMART CONTRACTS

Public Report

Oct 06, 2022

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 prepared by Verichains Lab on Oct 06, 2022. We would like to thank the Gomu 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 Gomu Smart 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 vulnerability in the given version of Gomu Smart Contracts.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Gomu Smart Contracts

Gomu mission is to help Web3 builders build the best experiences for their users by simplifying the creation of Web3 applications and services for every possible use case.

The Gomu Smart Contract is a collection and reward system that encourages the user to create liquidity pairs in the shape of Sudoswap.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Gomu Smart Contracts.

It was conducted on commit `1b1dd822ab707885e64ef29cc82b32aa2f7c1da8` from git repository *https://github.com/collectionswap/collectionswap*.

There are 3 files in our audit scope. They are `Collectionswap.sol`, `Collectionstaker.sol` and `RewardPoolETH.sol` files.

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 was 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

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The Gomu Smart Contracts was written in `Solidity` language, with the required version to be `^0.8.0`.

2.1.1. Collectionstaker contract

`Collectionstaker` is an initial contract that allows the caller to create a new `RewardPoolETH` corresponding with the `nft` address and `collectionSwap` contract. For each `RewardPoolETH`, only 5 ERC20 tokens can be set to the RewardToken list. Users can create `RewardPoolETH` to encourage other users to add pairs with a specific config (only pairs that have the same config with `RewardPoolToken` can be staked for profit).

2.1.2. Collectionswap contract

`Collectionswap` contract extends `ERC721Enumerable`, `OwnableWithTransferCallback` and `ERC1155Holder` contract. The user can create `SudoSwap` pair through this contract to receive an ERC721 token. The user can `stake` this token to the `RewardPoolETH` contract for reward Tokens. To avoid owner manipulation in the system, the owner of the pair is set to the `Collectionswap` contract and the user has some method to manage the pair through the ERC721 token. The user can `rescuse` `ERC20`, `ERC721` and `ERC1155` tokens stuck in pair. He also destroys the pair to withdraw all nft and native tokens from it.

2.1.3. RewardPoolETH contract

`RewardPoolETH` contract allows users to `stake` their pair tokens which are created by the `Collectionswap` to gain reward tokens. The workflow of this contract is similar to a Staking Contract, but the balance of user is based on the number of nfts and native tokens in his pair. The reward tokens are linearly released during the staking time, and the number of released tokens in the amount of time is fixed. It is split among the user following the rate of user balance in total staked balance. After the pool finishes 180 days, the deployer can withdraw remain reward tokens.

The contract implements the `rechargeRewardPool` function allows `deployer`/`protocolOwner` to recharge the reward pool. With this function, they can add new ERC20 reward tokens (the limit number of reward tokens is still `5`).

To `stake` pair tokens for profit, the pair env (`bondingCurve`, `delta`, `fee`, `nft`) must be similar to the value of `RewardPoolETH` which was set during the contract creation in the `Collectionstaker` contract.

Non-working pairs (user can't trade on it, balance of pair is less than the spot price) are also restricted to stake.

2.2. Findings

During the audit process, the audit team found no vulnerability in the given version of the Gomu Smart Contracts.

APPENDIX

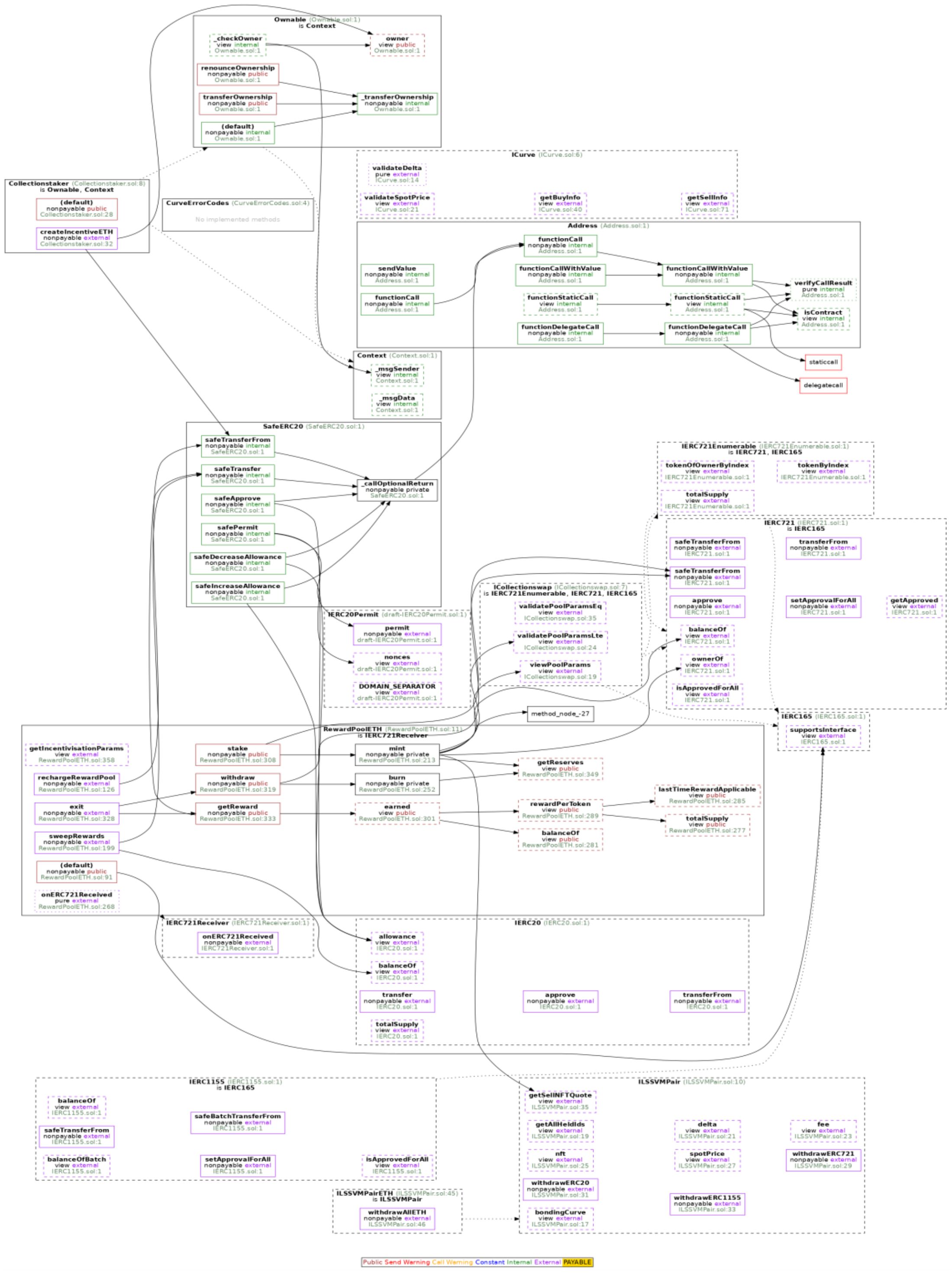

Image 1. Collectionstaker contract call graph

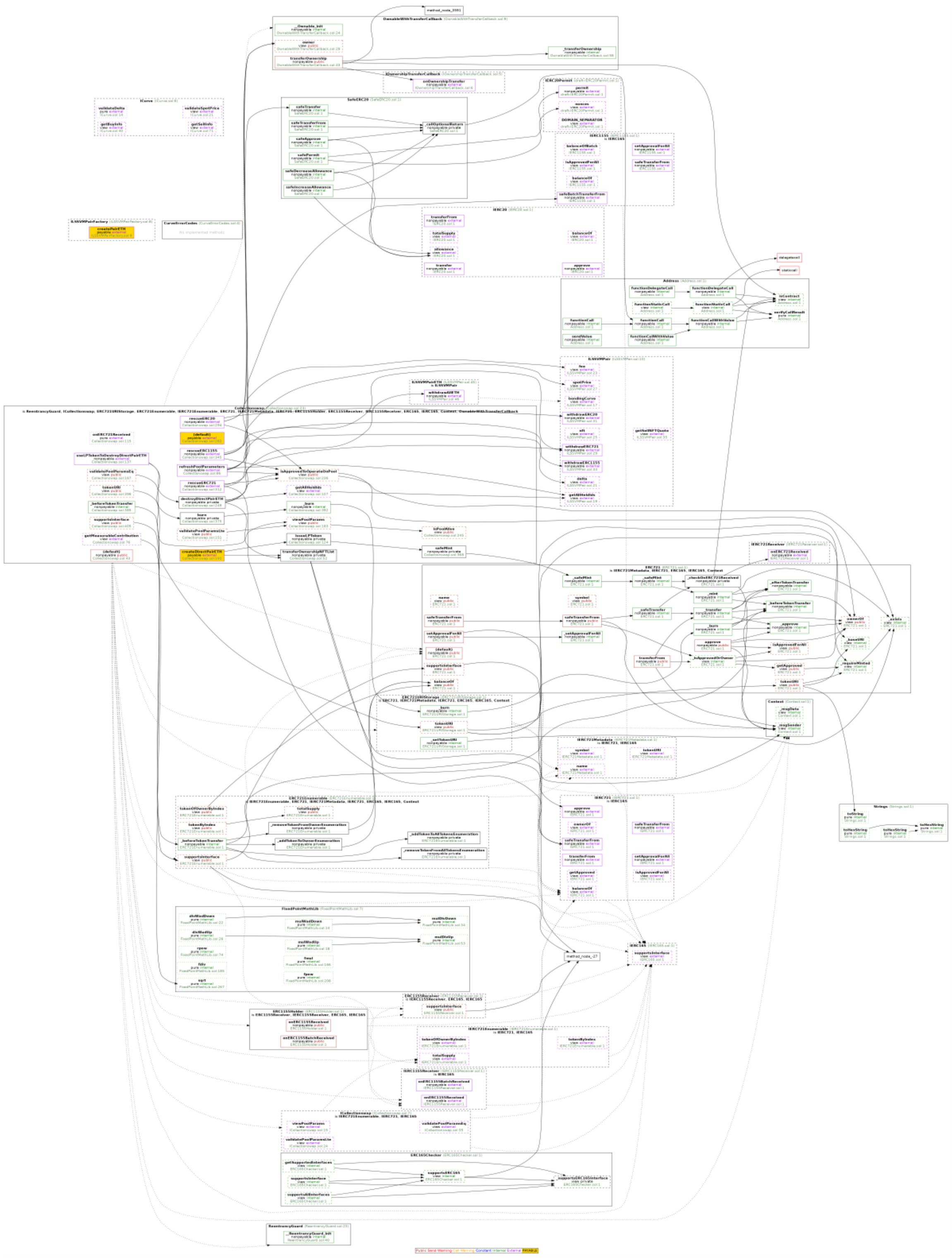

Image 2. Collectionswap contract call graph

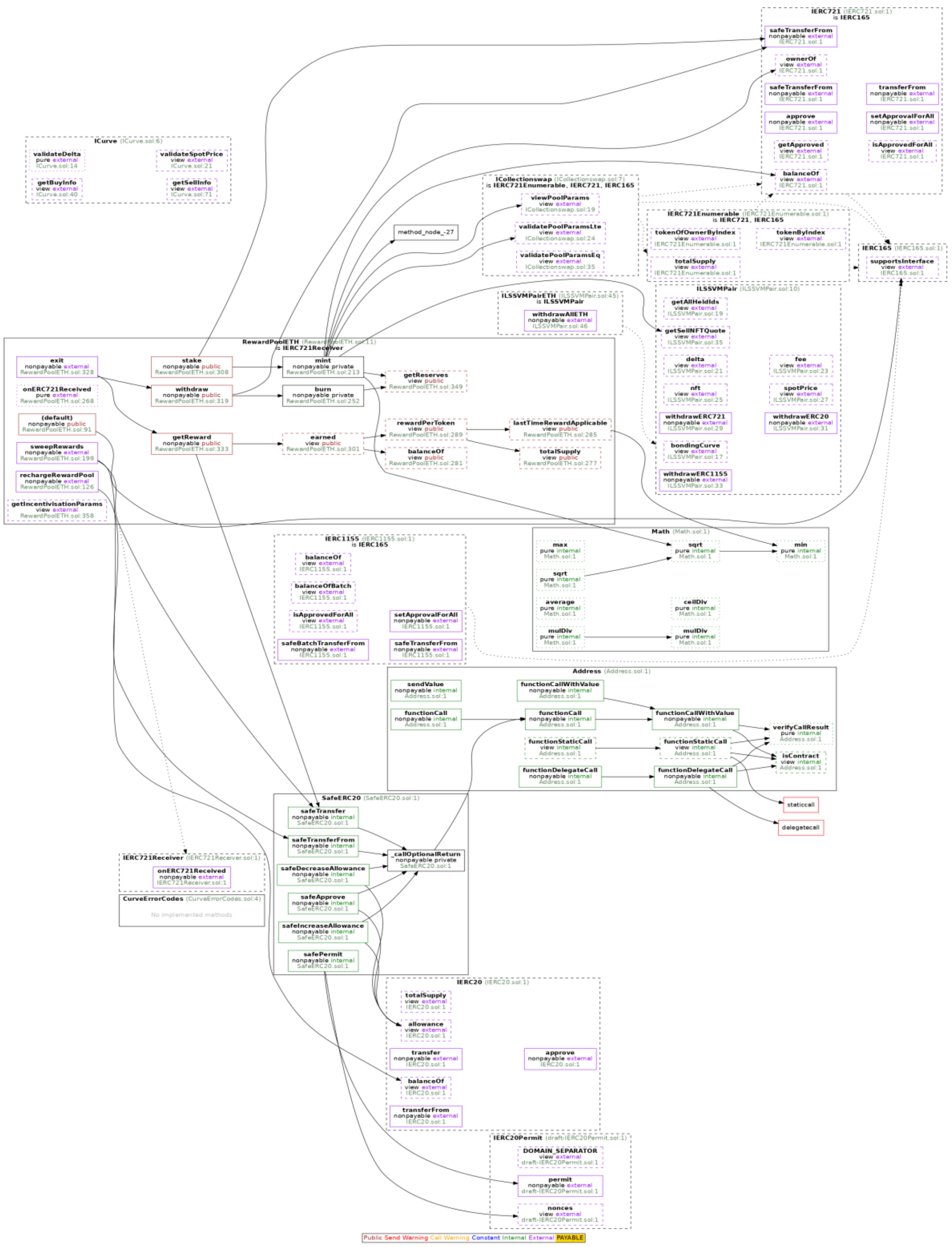

Image 3. RewardPoolETH contract call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Sep 28, 2022*	Public Report	Verichains Lab
1.1	*Sep 29, 2022*	Public Report	Verichains Lab
1.2	*Oct 06, 2022*	Public Report	Verichains Lab

Table 2. Report versions history