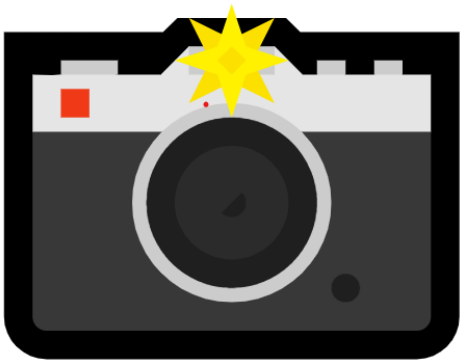
verichains

SECURITY AUDIT OF

ZKPHOTO SMART CONTRACTS



Public Report

May 12, 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 May 12, 2022. We would like to thank the ZKPhoto 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 ZKPhoto 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 had identified one vulnerable issue in the smart contract code. ZKPhoto team has resolved and updated the reported issue following our recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About ZKPhoto Smart Contracts

ZKPhoto (Private Authentic Photo Sharing) - this project demonstrates a small stepping stone toward #2 of Brian Gu's Six ZK Moonshots. The idea is to have an on-chain data marketplace where users can trade private data, for example, "a high-res image that downsamples to a known low-res image", using ZK. Within the scope of this proposal, the MVP is to implement a dApp that (0) use ZK to prove that the low-res image is downsized from an actual high-res image, (1) mint an NFT that contains the downsized image, as well as the hash of the original image, (2) implement an in-browser camera for authentic on-chain photo-taking, and (3) a marketplace providing secured transfer of the underlying full-res image.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of ZKPhoto Smart Contracts. It was conducted on commit f7d8e693672ed59a160ab24928e446c814403c6c from git repository *https://github.com/socathie/zkPhoto*.

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

SHA256 Sum	File
8ab3fa656fb714cbdcbb0a835d57525b372645f07890976f3e5ce614dbf610f1	**contracts/Base64.sol**
3ab3a90e76952e45f3528b7353cb028afcba2f4b0d52b5f87c18302383f0fd22	**contracts/zkPhoto.sol**
6f09ecbb77cc022898fa1d8e0d4074c414da61a7b7e96aea04c320397c8fe2e3	**circuits/zkPhoto.circom**
da952370fd9c953e02a816e91ae0ce14d0d74d205b9a8c0401823389f4ccf2f4	**circuits/util.circom**

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 ZKPhoto Smart Contracts was written in Solidity language, with the required version to be ^0.8.4. The source code was written based on OpenZeppelin's library.

There are two main parts in the audit scope as shown in the below section:

2.1.1. zkPhoto contract (ERC721 token)

The zkPhoto contract in zkPhoto.sol defines logic related to photo minting. According to the current implementation, users will need to input a high-resolution image which will be cropped and resized to 1024x1024. This image will be processed directly from their browser to generate the low-resolution image and a ZK proof (the client code can be referenced at *https://github.com/socathie/zkPhoto-ui*). The low-resolution image and ZK proof will be used as parameters for the mint function of the zkPhoto contract. The newly created NFT can be used to prove that the NFT owner is the one who owns the high-resolution image (the original image).

2.1.2. ZK circuits and the verifier contract

The ZK circuit for photo processing is defined in zkPhoto.circom, the original image of size 1024x1024x3 will be split into 16 images of size 256x256x3. Each image will be used as input for the zkPhoto circuit, which will produce an output image of size 16x16x3 and a Poseidon hash (multi-layer hashing of the original images). The circuit will then be compiled to R1CS which will be used to generate the verifier contract. This contract will be used to verify the low-resolution image and the proof before minting the NFT token.

Note: The security issues related to Circom and SnarkJS implementations are considered out-of-scope in this audit.

2.2. Findings

During the audit process, the audit team identified one vulnerability in the given version of ZKPhoto Smart Contracts. ZKPhoto fixed the code, according to Verichains's draft report, in commit 2cd9ca394a3ac4072bc101105f819da05b050f7e.

2.2.1. JSON injection inside token metadata LOW

When minting a new photo, the photo owner can input arbitrary data to the name, description, and image fields of the token metadata. However, these inputs may contain special characters like ", ; which can be combined to create a malicious JSON object.

```
function generateTokenURI(
    string calldata name,
    string calldata description,
    string calldata image
) private pure returns (string memory) {
    bytes memory dataURI = abi.encodePacked(
        "{",
        '"name": "',
        name,
        '", "description": "',
        description,
        '", "image": "',
        image,
        '", "external_url": "https://zkPhoto.one"',
        "}"
    );

    return
        string(
            abi.encodePacked(
                "data:application/json;base64,",
                Base64.encode(dataURI)
            )
        );
}
```

RECOMMENDATION

We should reject these special characters from the contract code. If not (for gas optimization), we must use the object from the token URI carefully (validate all properties before using them, reject unknown properties). In some cases, it may lead to prototype pollution vulnerability.

UPDATES

- *May 12, 2021*: This issue has been acknowledged and fixed by ZKPhoto team in commit 12b8d51632a6153c4e3b8799501bfe29fe48aff2.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*May 04, 2022*	Private Report	Verichains Lab
1.1	*May 12, 2022*	Public Report	Verichains Lab

Table 2. Report versions history