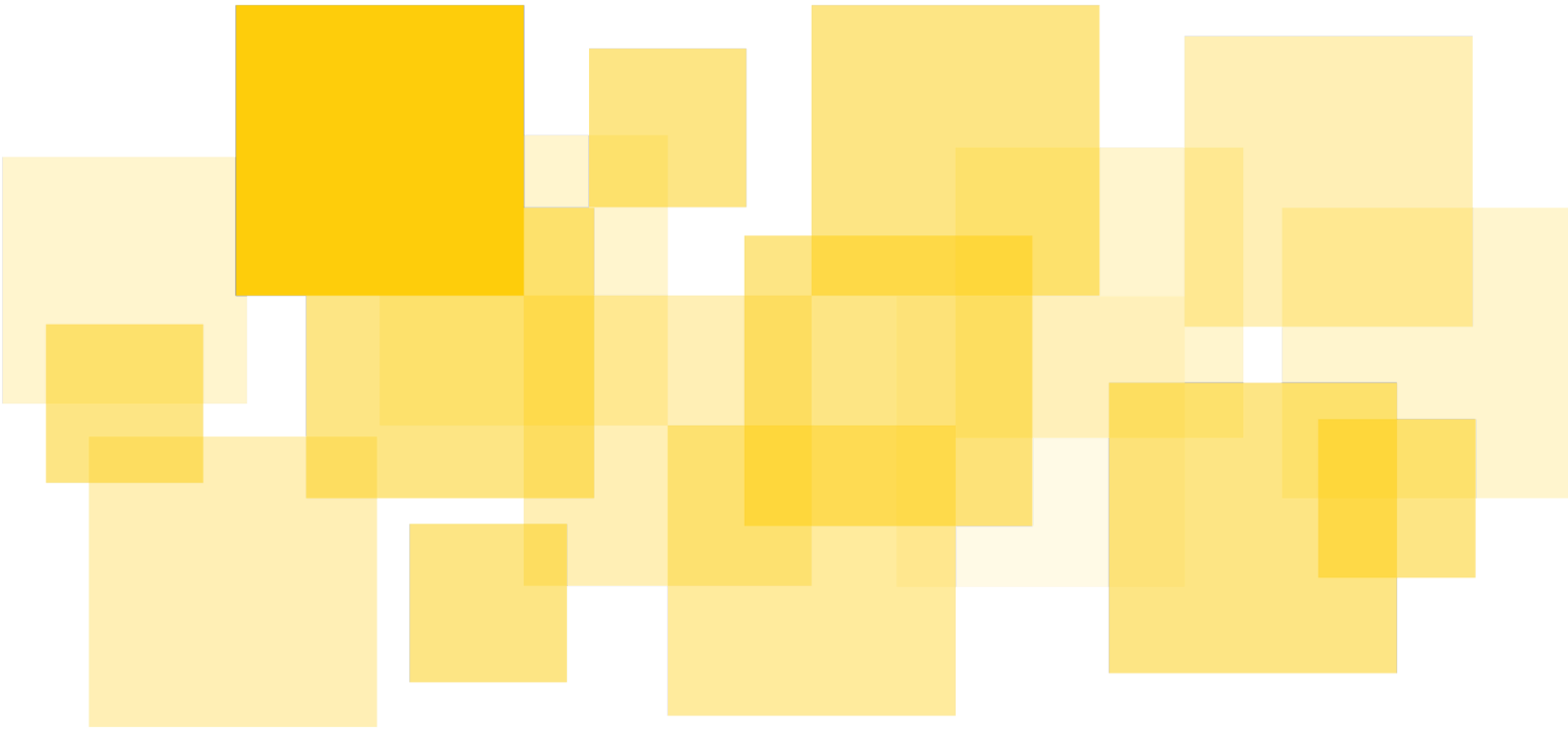
Security Audit Report

Jpg.Store V2 Contracts

June 10th, 2022



Prepared for Jpg.Store, Inc by

runtime verification

Table of Contents

Summary

Runtime Verification, Inc. acted as security code auditor on the jpg.store V2 smart contract. The review was conducted from May 04 to May 25, 2022. A follow-up code audit for the code revision, which is meant to address the issues found in the previous code review, was also conducted from May 26 to June 10, 2022.

JpgStore engaged Runtime Verification in checking the security of their v2 contract, which is an upgrade from their v1 contract. The v2 contract allows sellers to sell tokens or list tokens to accept offers from buyers. Buyers can buy tokens that are on sale or make offers to both listed tokens and unlisted tokens (in someone's wallet) . The token owner can accept/reject/ignore offers. Buyers are able to cancel their sell before the token is bought or cancel the listing before accepting the offer. The buyer is able to cancel his offers before the offer is accepted. The onchain contract acts as an escrow for the tokens on sale or on listing and the Ada in an offer. The onchain contract needs to ensure correct payouts to different parties when the escrowed tokens and ada are bought or canceled.

The onchain contract is quite flexible that anyone can use this contract to escrow any sells or offers. The jpg.store web server (can be viewed as the off-chain part of the contract) ensures that only eligible sales or offers on the onchain contract are synchronized and shown on the website. Based on this design we **strongly** recommend users (buyers or sellers) transact through the official website, otherwise you may encounter unexpected losses. We also recommend users to check the transaction details carefully before signing a transaction while using the DApp.

Introduction

Runtime Verification engaged in the code audit for Jpg.Store onchain contract and a follow-up audit to review the changes.

Scope

The code audit phase focuses on reviewing one code repository assisted with the frontend and backend of the NFT marketplace code repositories. The frontend and backend repositories serve to provide a better understanding of the onchain contract and are ***not*** part of this engagement. Although we did find several issues on the backend when it interacts with the onchain contract. The code review is conducted on *source code* of the commit hash **a603e3ebb837219799f1596a7b0408e3ed6df297** of the contracts repo. The following source code files in this commit hash are included in the review:

- src/Canonical/Shared.hs
- src/Canonical/DebugUtilities.h

- src/Canonical/JpgStore/BulkPurchase.hs

The review is limited to only the above source files which are the onchain code of the contracts. Any other artifacts such as testing or deployment scripts in the repository are not part of this review.

The purpose of the follow-up code audit is to make sure that all the issues found in the code audit phase have been addressed. During the main audit, we found several issues that can cause potential vulnerabilities for the system as a whole and the jpg.store team decided to fix those issues in the marketplace application. Thus there is no code change of the on-chain contract . In the follow-up audit phase, the auditor reviewed the pull requests for the issues found by the auditor in the code audit phase (findings A1-A3). The auditor also helped to review pull requests for the marketplace application, these pull requests are meant to address the issues found during the code audit phase.

Protocol Configurations

There is no special configuration for the smart contracts and Runtime Verification independently verified that the source code given in the above commit is compiled (with the compiler configuration in the repo) to an onchain plutus code with script hash: addr1wxteexw88ytjvuh55qhhzd2w9q4h0mljn0jevhmxcce64ds69mn6y

Assumptions

All the code reviews are based on the following assumptions and trust model:

1. We assume the offchain code which interacts with the onchain contract is unsafe.
2. This code repository is based on the V1 of Plutus API. We also assume the correctness of the Plutus platform. In particular, we assume that the Plutus Core compiler works correctly by compiling the Haskell code to the Plutus Core code.
3. The NFT minting policy shall ensure that the NFT is unique

Methodology

We conduct the code review manually. Although manual code review cannot guarantee to find all possible security vulnerabilities as mentioned in Disclaimer, we have followed the following approaches to make our review as thorough as possible. First, we rigorously reasoned about the business logic of the contracts, validating security-critical properties to ensure the absence of loopholes in the business logic. Second, we carefully modeled the design in pseudo code so that we can find details which may be missing from the specification. Finally, we modeled the dependency (token dependency for transactions and token minting) among the contracts; tried out all possible combinations of eUTXOs to construct desired and undesired transactions; checked the following categories of vulnerabilities for each possible transaction:

1. Token related issues. Tokens are assets and are the means to share data (together with the datum) and to achieve access control in Plutus contracts. So the correctness of minting and usage of the tokens and their datum is paramount for the correct functioning of Plutus contracts.
2. Plutus specific implementation issues. There are semantic gaps between the source code language (Haskell) and the onchain assembly code (Plutus Core). It is easy for developers to ignore issues that come inherently in the Plutus platform.
3. Marketplace related properties.

We considered the following attack surfaces for token related issues:

1. Fake token attacks
2. Unsound minting policies
3. Side minting attacks
4. Token uniqueness attacks
5. Violation of single transaction execution
6. Contract clone attacks
7. Deadly transaction repetition

All the issues have been reported to and are either acknowledged or addressed by SundaeSwap Labs.

Severity Classification

The following bug severity classification system is used in the report:

- P1: Highest priority; loss of funds, deadlock of funds, hijack of protocol, arbitrary minting of tokens, etc.
- P2: Severe disruption, DDOS, etc. No funds are technically lost, but makes the protocol unusable for long periods of time
- P3: Inconvenient, or non-intuitive behavior of the protocol; loss of funds through users fault, but where the mistake is very easy to make; etc.
- P4: Code organization, performance, clarity; no major / logical impact on the protocol, but technically changes the behavior
- P5: Purely documentation / cosmetic, like wording in comments or error messages, etc.

Disclaimer

provided or is otherwise publicly available, there are likely additional unknown risks which otherwise exist. This report is also not comprehensive in scope, excluding a number of components critical to the correct operation of this system. This report is for informational purposes only and is provided on an "as-is" basis and you acknowledge and agree that you are making use of this report and the information contained herein at your own risk. The preparers of this report make no representations or warranties of any kind, either express or implied, regarding the information in or the use of this report and shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein.

List of Findings

The following list of findings consists of issues identified by the auditor from Runtime Verification (A1-A3).

A1. Negative values in payout may cause token buyers to pay more than the listing price

In Jpg.Store when a seller lists a token for sale, then a fixed percentage of the listing price would be sent to the seller when the token is bought by a buyer. The remaining of the prices would be paid to the token creator as royalties and marketplace fee. The buyer would pay the amount of the listed price of ADA to buy the token.

However, a malicious seller could then update the list by using a negative value in the payout to marketplace fee or royalties which makes listing price lower and attractive to the potential buyer. However, the onchain contract uses absolute values and the negative value would be treated as positive value. If the buyer does not check the buy transaction details carefully before signing the transaction, the buyer needs to pay more than the listing price.

Severity: P1

Recommendation

We strongly recommend Jpg.Store users always check the transaction details before signing it to make sure that the token and ada transfers in the transaction are expected, otherwise refuse to sign.

For Jpg.Store, make sure that no negative values are presented in the payout data.

Status: Addressed by client. See Pull Request 193 on the backend server.

A2. The buyer of a listed token may not receive the token.

Description

A malicious seller of a token first lists the token on the jpg.store page. Then the seller updates the payout of the sell eUTXO by adding a payout which sends the listed token to an address controlled by the seller, without using the jpg.store webpage. The buyer of the listed token may not receive the listed token if he/she does not carefully check the buy transaction details when signing the transaction. In this way, the token will be sent to the seller's address, not to the address specified by the buyer.

Severity: P1

Recommendation

The best way to fix this would be through the on-chain code by checking that there is an output eUTXO in the buy transaction which sends the listed token to the transaction signer's address.

However, Jpg.Store decided to fix the issue through the marketplace application such that an output eUTXO is added when building the buy transaction.

We strongly recommend Jpg.Store users always check the transaction details before signing it to make sure that the token and ada transfers in the transaction are expected, otherwise refuse to sign.

Status: Addressed by client. See Pull Request 193 on the backend server.

A3. The seller may not get the fair amount of Ada for the sell

In Jpg.store, an offeror can make offers to an unlisted token with the amount of ADA he/she would want to buy the token. Then if the price seems reasonable, the owner of

the desired token can accept the offer. The owner would expect to accept a fixed percentage of the offered price, the remaining amount is paid to the creator of the NFT as royalties and a fixed percentage is paid as marketplace fee.

A malicious offerer could first make an offer through the Jpg.Store with correct payments of royalties to the NFT creator and the marketplace fees. Then the offeror can modify the offer, without using the Jpg.Store Dapp, by adjusting the amount of Ada paid to as royalties and marketplace fee such that the offered price is the same but only a little amount is paid to the owner. Then when the owner accepts the offer but does not check the transaction details carefully, then the owner of the token will only get much less Ada than the offered price.

Severity: P1

Recommendation

We strongly recommend Jpg.Store users always check the transaction details before signing it to make sure that the token and ada transfers in the transaction are expected, otherwise refuse to sign.

The recommended fix for the problem is that when synchronizing the updated offers Jpg.Store shall check the validity of the modified order such that the owner gets a fixed percentage (as specified by the transaction rules) of the offer price.

Status: Addressed by client. See Pull Request 193 on the backend server.