STAUBLI
SOFTWARE
SOLUTIONS

Smart Contract Code Review and Audit Report

July 27th, 2023

—

Created for: 0xShuffle

Roger Staubli
Staubli-Software-Solutions
Sonnhaldenstrasse 4a
6122 Menznau, Switzerland

STAUBLI SOFTWARE SOLUTIONS

Document

Name	Smart Contract Code Review and Audit Report
Carried out by	Roger Staubli | Founder Staubli-Software-Solutions
Language	Solidity
Methods	Best Practice Review, Manual Code inspection, Automated Code inspection
Repository	https://bitbucket.org/DrenImeraj/shuffle-smart-contracts
Commit	f4a394a1940a1262a71944f465c088d0e70bc855
Technical Documentation	Yes: Provided Code Comments
Unit Tests	Yes
Timeline	22.06.2023 – 27.07.2023
Changelog	30.06.2023 – Initial Audit 27.07.2023 – Final Audit

STAUBLI SOFTWARE SOLUTIONS

Executive Summary

After the changes from the initial audit, the final audit resulted in 0 critical severity issues, 0 medium severity issues, and 0 low severity issues. In addition, 1 informational suggestions were provided.

0xShuffle is a marketplace designed for hosting NFT raffles, where users can list their ERC721 or ERC1155 NFTs for sale. Sellers can specify the number of raffle tickets available for purchase and set the cost per ticket. Participants can then buy tickets for a chance to win the listed NFTs. At the conclusion of the raffle, either when all tickets are sold or the listing time expires, a winner is selected using Chainlink VRF (Verifiable Random Function) to ensure fairness. The winning participant receives the NFT, and the proceeds from ticket sales are transferred to the seller.

Overall, the contracts are well programmed, with a clear separation of concerns and respect for Smart Contract best practices. Also, the test coverage is exceptionally high, with thorough coverage of both normal and edge cases.

Scope

The audit contained all Smart Contracts, tests, and deployment scripts from the specific commit described on the second page. Well-known dependencies like OpenZeppelin implementations were out of scope.

The following Smart Contract files were reviewed:

a58fced310f63bb767892128b4be50c004a84bc6	contracts/ChainlinkVRF.sol
6df3b61c76d3aed59a16e3d2ac254b76cafdbe73	contracts/Managed.sol
0f36c0afd7499026d02d88fd87a045d57c0c7de8	contracts/Shuffle.sol
2245fa3fba31ea2589d061d9bc24ca08394a36b3	contracts/Interfaces/IManaged.sol
ecb101772b3651fded6f3432a4790b5ce1e0d83f	contracts/Interfaces/IShuffle.sol
7e7924643a8bdf0a5d3707fd7f11c4d489327c53	contracts/Interfaces/VRFConsumerBaseV2Upgradeable.sol
d962dc194fd295ed663896b62825c524e8671834	contracts/Interfaces/VRFCoordinatorV2Interface.sol

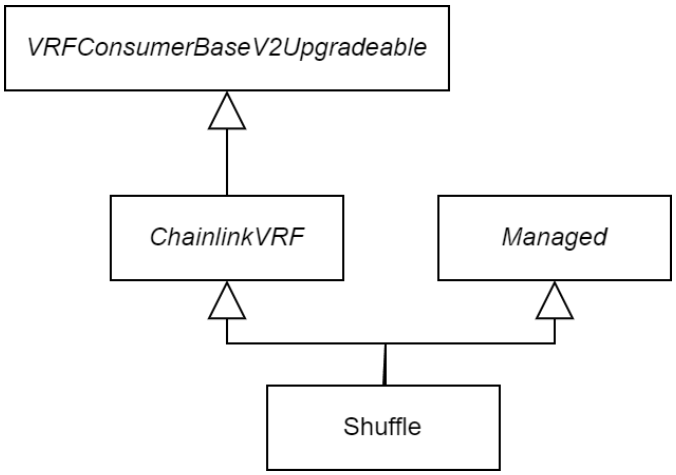
The following tests and deployment scripts were reviewed:

d60dd5f2d9dcdfb06055734a54306993abf05244	scripts/prod/constants.ts
a02acbdd5c4072edfea611d77c757ec95da260ef	scripts/prod/deploy.ts
72608b00230a014ae9f8b165841a2c79eae853af	scripts/dev/constants.ts
a02acbdd5c4072edfea611d77c757ec95da260ef	scripts/dev/deploy.ts

41bb969246c58d4c1260b3968f2785ef60181861	scripts/dev/upgrade_shuffle.ts
a027c61894a951bd1715622198152036553eae34	test/unit/Managed.ts
a85959a05f224315efa612d85c08c4bd42c56111	test/unit/Shuffle.ts

System Description

The 0xShuffle contracts contain abstract Chainlink VRF contracts, which handle the generation of Verifiable Random Numbers. On the other hand, they contain an abstract Managed contract, which handles the configuration of the main shuffle contract, by setting global configuration parameters and providing internal helper functions. Finally, the Shuffle contract is the only deployable contract, which combines all the abstract contracts and implements the main business logic of the 0xShuffle ecosystem.



VRFConsumerBaseV2Upgradeable.sol

The contract "VRFConsumerBaseV2Upgradeable" is an interface for contracts that utilize Chainlink Verifiable Random Function (VRF) randomness. It enables contracts to receive and process random values from an external oracle in a verifiable manner. Contracts inheriting from this interface need to implement the "fulfillRandomWords" function to handle the VRF response. The "rawFulfillRandomWords" function is called by the VRFCoordinator contract to validate and fulfill the randomness request.

Chainlink.sol

The "ChainlinkVRF" contract is an abstract contract that integrates with the Chainlink Verifiable Random Function (VRF) system. It inherits from the "VRFConsumerBaseV2Upgradeable" contract, which provides the interface for interacting with the VRF system. The contract includes functions to request random seeds for specific listings and handle the fulfillment of those requests. It utilizes the "VRFCoordinatorV2Interface" contract to communicate with the

Chainlink VRF service. The contract allows for the configuration of key parameters such as the VRF coordinator, key hash, subscription ID, callback gas limit, and request confirmations.

Additionally, it provides a function to load the obtained random seed for a specific listing. Contracts inheriting from "ChainlinkVRF" are expected to implement the "loadListingSeed" function to define the logic for handling the obtained seed.

Overall, the "ChainlinkVRF" contract acts as a bridge between the Shuffle.sol contract and the Chainlink VRF system, enabling the generation and utilization of verifiable random numbers in a secure and transparent manner.

Managed.sol

The "Managed" contract is an abstract contract that holds and manages global fields used by the Shuffle contract. It provides functionalities for enabling or disabling contracts, managing allowed collections, setting commission-related parameters, defining limits for listing duration and ticket count, collecting listing fees, transferring NFTs between the users and the contract, and transferring ETH. It also implements access control using OpenZeppelin's AccessControlUpgradeable contract and supports ERC721 and ERC1155 token standards. This contract is not deployable on its own and serves as a base contract for the main Shuffle.sol contract.

Shuffle.sol

The “Shuffle” contract is the deployable contract of the ecosystem. The contract combines all the necessary abstract contracts and adds the business logic. The shuffle contract allows a user to list an NFT and set listing parameters like the ticket price, the ticket count, the opening time or the duration of the listing. While listing an NFT, the NFT is transferred to the Shuffle contract and global variables are set for the listing. After that users can start to buy tickets for the listing to be part of the lottery to win the NFT. A lister is able to cancel the listing if no tickets have been sold. In addition, a moderator can also cancel the listing even though tickets have been sold. When canceling the listing, the NFT is transferred back to the lister and the bought tickets can be refunded by the users. If the duration of the listing is finished or all the tickets are sold, the listing goes into the evaluation stage of the winner. This means that the VRF random number is requested. When the VRF oracle returns the random seed, the winner is determined.

To determine the winner, the random seed is calculated modulo the total ticket counts. It results in a random number between 0 and the total ticket counts. With binary search, the user is found which holds the slot in the chronologically ordered bought ticket array.

The winner will receive the NFT, the sales commission is transferred to the commission collector, the listing costs are collected and the collected ether are sent to the lister. If not all tickets were sold, the unsold tickets belong to the lister. Hence, there is a chance that the lister will win back the NFT and collect the collected ether.

Best Practices

The Best Practices analysis does not cover direct vulnerabilities. It shows the overall project and code structure and if best practices were applied. This is a good measurement to validate the audit result, as a clean and understandable code base indicates that most bugs and vulnerabilities were found.

☑	The code was provided in a source control.
☑	A technical documentation was provided (Yes, inline NatSpec documentation).
☑	Minimal code duplication.
☑	Smart Contracts are unflattened.
☑	A recent solidity version was used (0.8.14).
☑	A framework for testing and deployment was used (Hardhat).
☑	There are tests.
☑	Tests are easy to run.
☑	There is no unused code.
☑	The code follows standard Solidity naming conventions.

The developers applied all best practices in the following code base. The code base contains unit tests for 100% line coverage, and the contracts and functions are well-commented.

Findings

The findings were categorized into the following four different levels:

- **Critical:** Potential loss of funds is expected. They need to be fixed immediately.
- **Medium:** Errors that can cause the contracts to fail. Manual changing needs to be done to restore the contract functionality.
- **Low:** Errors that can cause the contracts to fail in specific conditions like edge cases.
- **Informational:** Suggested improvements of the contracts that do not have security-related issues (e.g. gas optimization).

Critical

No Critical issues were found

Medium

No Medium issues were found

Low

IC1: Initializer call missing

In `contracts/Shuffle.sol` initializers for the reentrancy guard are missing. It is best practice to initialize all inherited initializers even if they do not implement essential logic.

Recommendation

Add `__ReentrancyGuard_init();` to the `initialize()` function of the mentioned contracts.

Status

Resolved in commit: 96cbdbc0793cbaf3e11add02416eba986a424679

RE1: Reentrancy vulnerability

In `contracts/Shuffle.sol` the `list()` function contains a minor reentrancy vulnerability. The `_pullNFT` function calls an untrusted external contract (NFT contract). The function can be reentered and the `listingCount` can be artificially increased without a listing. This does not have any impact on the contracts functionality, could however mess up logic implemented by clients.

Recommendation

Add the `nonReentrant` modifier to the `list` function.

Status

Resolved in commit: ba748c965e2ddf9cb070a4266df2f889634e1deb

EC1: Events Missing

It is recommended that functions, which change the global state of a smart contract, emit events such that changes can easily be tracked on the blockchain. The following functions miss events:

`contracts/Shuffle.sol: updateVrfConfiguration`

`contracts/Managed.sol: setContractEnabled, setAllowedCollectionsOnly, setCommissionCollectorAddress, setSaleCommission, setOpenTimeLimits, setMaxListingValue, setListingCost, collectListingFees`

Recommendation

Provide events for the listed functions.

Status

Resolved in commit: e4ccf38ce46135e7043f8db9f6c8b461d1592131

IM1: Initializer modifier missing

The following contracts contain an internal Initializer method but do not contain the `onlyInitializing` modifier. Therefore, the initializer methods could potentially be called multiple times:

`contracts/ChainlinkVRF.sol: __ChainlinkVRF_init`

`contracts/Interfaces/VRFConsumerBaseV2Upgradeable.sol: __VRFConsumerBaseV2Upgradeable_init`

Recommendation

Import `@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol` in the contracts and add `onlyInitializing` modifier to the initializer functions.

The `__VRFConsumerBaseV2Upgradeable_init` method is called twice by the `ChainlinkVRF` contract. Consider to use an additional setter function there, to update the `VRFCoordinator`

Status

Resolved in Commit: 04e131324468c46ab3cdbdfed8797f39f200a69a

IR1: Inappropriate receiver of Funds

In the contract `contracts/Managed.sol,` in the function `collectListingFees(),` the receiver of the listing fees is the `msg.sender`. As the message sender can be everyone that has `UPDATER_ROLE` on the contract, the funds might be sent to an inappropriate/malicious receiver. The `UPDATER_ROLE` should be reserved for only configuration changes (which can be set back) and not to send funds to.

Recommendation

Consider sending the listing fees to a defined and secure treasury account.

Status

Resolved in commit: d33d0debdb06299f96c593799c96df035a017563

Informational

EF1: Set public functions to external

Functions that are only called from external sources should be declared external to optimize gas costs. The following functions are visible as public but are not accessed within the contract:

`contracts/Managed.sol: setContractEnabled, setAllowedCollectionsOnly, setAllowedCollection, setCollectionStandard, setCommissionCollectorAddress, setSaleCommission, setOpenTimeLimits, setTicketCountLimits, setMaxListingValue, setListingCost, collectListingFees,`

`contracts/Shuffle.sol: list, buy, cancelListing, moderatorCancelListing, refund`

Recommendation

Change the functions from `public` to `external`

Status

Resolved in commit: 3ddd6c983ae576173f64f7cf715ae3ff87e3346d

UC1: Unnecessary check

In `contracts/Managed.sol` on line `76` is an unnecessary check. If the first condition is false (in this case, `allowedCollectionsOnly` is true), the `allowedCollectionsOnly` check to be true is not necessary.

Recommendation

Change the require statement to:

```
require(!allowedCollectionsOnly || allowedCollection[collection], "Collection not in allowlist");
```

Status

Resolved in commit: f4a394a1940a1262a71944f465c088d0e70bc855

NS1: Non-Standard Implementation

The contract `contracts/Interfaces/VRFConsumerBaseV2Upgradeable.sol` is a non-standard implementation from Chainlink.

Recommendation

Consider using the standard implementation from Chainlink found here:

https://github.com/smartcontractkit/chainlink/blob/develop/contracts/src/v0.8/dev/VRFConsumerBaseV2Upgradeable.sol

Status

Accepted

Limitation

This code review was conducted carefully and on a best-effort basis. However, this does not guarantee that there are any undiscovered issues and vulnerabilities. This audit gives no warranties on the security of the code.