Audit Report

Archisinal Marketplace

v1.0

August 21, 2024

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Archisinal Technologies Limited to perform a security audit of Archisinal Marketplace.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/Archisinal/marketplace-contracts
Commit	`bfb5ced65e7aa9b3ccf6474bbb8e6a5e9e3dff3f`
Scope	All contracts were in scope.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Archisinal Marketplace is an NFT marketplace for the Astar Network, where creators can monetize their designs while retaining full ownership of them. The marketplace allows NFTs to be traded through secondary sales with support for creator royalties.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase’s code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Low-Medium**	-
Code readability and clarity	**Medium-High**	Most functions and state variables are documented with clear and concise comments.
Level of documentation	**Medium-High**	Documentation and diagrams are available in the `README` file.
Test coverage	-	End-to-end tests are provided in the `test` folder.

Summary of Findings

No	Description	Severity	Status
1	Owner loses their NFT if the auction ends without bids	Critical	Acknowledged
2	Attackers can back-run allowance to buy NFT at a low price, causing a loss for the owner	Critical	Acknowledged
3	Native funds are incorrectly transferred to the caller	Critical	Acknowledged
4	Collection ownership renouncement leads to locked assets	Major	Acknowledged
5	Bidders cannot send native funds to bid on NFTs	Major	Acknowledged
6	Incorrect logic condition check in the `_admin_or_owner` function	Major	Acknowledged
7	Lack of validation for royalty leads to stuck funds	Major	Acknowledged
8	Incorrect creator accounts implementation	Major	Acknowledged
9	Excess funds sent by users will be stuck	Minor	Acknowledged
10	Irrecoverable protocol functionalities due to ownership transfer and renouncement	Minor	Acknowledged
11	Role-based access control assignment does not follow best practices and centralization issues	Minor	Acknowledged
12	Identical token IDs may lead to confusion	Minor	Acknowledged
13	Redundant validations	Informational	Acknowledged
14	NFT can be listed as free	Informational	Acknowledged
15	Unnecessary `account_id` function	Informational	Acknowledged
16	Code optimizations	Informational	Acknowledged
17	Missing validation on strings may lead to cross-site scripting	Informational	Acknowledged
18	Unused errors	Informational	Acknowledged

Detailed Findings

1. Owner loses their NFT if the auction ends without bids

Severity: Critical

The `claim_nft` function in `impls/auction/mod.rs:254` is responsible for distributing the NFT and funds when an auction ends. If there are bids, the NFT is sent to the highest bidder, while the tokens are sent to the auction creator.

However, if there are no bids for the auction, the NFT is not refunded to the owner. This causes a loss for the owner, as the NFT should be refunded if the auction fails.

Recommendation

We recommend refunding the NFT to the owner if there are no bidders for the auction.

Status: Acknowledged

2. Attackers can back-run allowance to buy NFT at a low price, causing a loss for the owner

Severity: Critical

To list an NFT for sale or auction, the owner must perform two transactions. The first transaction approves an allowance for the `marketplace` contract, and the second transaction calls the `list_nft_for_sale` or `list_nft_for_auction` functions in `contracts/marketplace/lib.rs:244` or `283`.

While both functions correctly validate that the `creator` owns the NFT in `impls/marketplace/mod.rs:52-54` and `impls/auction/mod.rs:59-61`, they do not validate that the caller is the owner. This means anyone can list or auction the NFT for sale if the owner approves a valid allowance.

This opens up an attack vector where an attacker can back-run the allowance transaction to list the NFT with a low price (typically under market value) and purchase it. Back-running means that the attacker's transaction is executed immediately after the victim's transaction is processed.

An example attack scenario is illustrated below:

1. The owner approves an allowance for the `marketplace` contract.
2. The attacker back-runs the transaction to list the NFT with a low price.
3. The attacker purchases the NFT at a discounted price.
4. The owner is forced to sell their NFT at a loss.

5. The attacker gains profit after selling the NFT at its market price.

Although both the `list_nft_for_sale` and `list_nft_for_auction` functions are susceptible to this vulnerability, sophisticated attackers would use `list_nft_for_sale`, as it allows them to purchase the NFT at a lower price immediately before the owner or admin cancels the trade.

Recommendation

We recommend modifying the `list_nft_for_sale` and `list_nft_for_auction` functions so only the owner can list or auction their NFTs.

Status: Acknowledged

3. Native funds are incorrectly transferred to the caller

Severity: Critical

In `impls/shared/currency.rs:88`, the `transfer` function distributes native funds to the caller. This is incorrect because the recipient should be the `to` parameter instead of the caller.

Consequently, bidders who were outbid will not receive their refunds in `impls/auction/mod.rs:236`, and the auction creator and collection owner will not receive the winning bid and royalties in lines `303-308`, causing a loss of funds.

Recommendation

We recommend modifying the `transfer` function to distribute the native funds to the correct recipient.

Status: Acknowledged

4. Collection ownership renouncement leads to locked assets

Severity: Major

Upon a successful NFT trade, the `marketplace` contract distributes royalty fees to the collection owner.

An issue occurs though when the `claim_nft` function in `impls/auction/mod.rs:305-306` returns a `CollectionOwnerNotFound` error if the collection owner renounces ownership. This would cause the function to always return an error, preventing the auction from being concluded.

Consequently, the NFT and bidder funds will be stuck in the contract, causing a loss for the winning bidder and the auction creator.

Additionally, the `buy_nft` function in `impls/marketplace/mod.rs:161-163` only distributes royalties if the collection owner exists. However, the royalties will be stuck in the `marketplace` contract if the ownership is renounced.

Recommendation

We recommend sending the royalties to the creator if the collection ownership is renounced.

Status: Acknowledged

5. Bidders cannot send native funds to bid on NFTs

Severity: Major

In `contracts/marketplace/lib.rs:298`, the `bid_nft` function is not annotated with the `#[ink(payable)]` attribute. This is problematic because bidders are expected to send native tokens for `Currency::Native` auctions in `impls/auction/mod.rs:233` as part of the bidding process.

Consequently, auctions that use native tokens as the medium of exchange will receive no bids, breaking the marketplace's functionality.

Recommendation

We recommend annotating the `bid_nft` function with the `#[ink(payable)]` attribute.

Status: Acknowledged

6. Incorrect logic condition check in the `_admin_or_owner` function

Severity: Major

In `impls/admin_access/impls.rs:46-49`, the `_admin_or_owner` function validates whether the caller is one of the admins or is the contract owner. However, the logic in line `47` checks that the caller is both the admin and the owner, which differs from the intended functionality.

Since there can only be one contract owner, admins who are granted roles from the contract owner cannot call the privileged functions as expected.

Recommendation

We recommend modifying the condition checks to use AND (&&) comparison.

Status: Acknowledged

7. Lack of validation for royalty leads to stuck funds

Severity: Major

The `arch_nft` contract defines a `new_default` constructor in `contracts/arch_nft/lib.rs:120`. The issue is that the constructor does not validate the royalty value to be less than the max BPS value of `10000` (see lines `96-98`).

This is problematic because a royalty value larger than the max BPS value breaks the marketplace's functionality. Specifically, the `apply_fee` function in `impls/shared/utils.rs:22-30` computes the royalty fees when concluding a trade. If the fee value exceeds `10000`, the `IntegerUnderflow` error in line `30` will be triggered.

Consequently, the `claim_nft` function will error in `impls/auction/mod.rs:297`, reverting the transaction. This will result in the NFT and bidder's funds being stuck in the contract, causing a loss of funds for the auction creator and the winning bidder.

Recommendation

We recommend validating the royalty value to be less than `10000`, similar to `contracts/arch_nft/lib.rs:96-98`.

Status: Acknowledged

8. Incorrect creator accounts implementation

Severity: Major

The `account_manager` contract in `contracts/account_manager/lib.rs:105-131` allows users to create `User` and `Creator` accounts. Their implementations can be found in the `impls/user` and `impls/creator` directories.

However, the `Creator` account contract defined in `contracts/creator/lib.rs` implements the same functionalities as the `User` account contract in `contracts/user/lib.rs`, which is incorrect.

Consequently, the `create_creator_account` function will incorrectly create `Creator` accounts as `User` accounts, causing creators to be unable to create collections and breaking the protocol's functionality.

Recommendation

We recommend adding an appropriate implementation for the `Creator` contract. Namely, the `Creator` and `CreatorImpl` traits from `impls/creator/impls.rs` should be implemented.

Status: Acknowledged

9. Excess funds sent by users will be stuck

Severity: Minor

The `marketplace` contract allows sellers to decide whether they want to receive native or PSP22 tokens as the medium of exchange. When purchasing or bidding on the NFTs, the `assure_transfer` function in `impls/shared/currency.rs:126-137` implements validation mechanisms to ensure the user sends sufficient native funds.

However, no logic has been implemented to refund the user if they have sent excess funds. If a user sends more funds than required, those tokens will be stuck in the contract. This can also happen when the traded currency is PSP22 tokens, but the user included native tokens when calling the functions.

The affected functions are `buy_nft`, `buy_batch`, and `bid_nft` in `impls/marketplace/mod.rs:123`, `178`, and `impls/auction/mod.rs:194`, respectively.

While blockchains generally do not protect users from sending funds to the wrong accounts, reverting extra funds increases the user experience.

Recommendation

We recommend refunding the caller if they sent excess native tokens.

Status: Acknowledged

10. Irrecoverable protocol functionalities due to ownership transfer and renouncement

Severity: Minor

The `account_manager` contract allows users to create accounts as individual contracts in `contracts/account_manager/lib.rs:105-131`. The contracts are instantiated with a hard-coded salt and then added to the mapping with the key entry as the caller's address. This approach limits the caller from creating only one instance of the `User` and `Creator` account types, as the `AccountAlreadyExists` error will be raised if the account already exists.

The instantiated contracts implement an `Ownable` trait, allowing functionalities to transfer and renounce ownership. If the user wants to create a new account type (e.g., because contract ownership was transferred or renounced by mistake), the transaction will fail.

Recommendation

We recommend overriding the `Ownable` trait functions to return an error when the user transfers or renounces contract ownership. If the ownership transfer is a required feature, the transfer entry point should be implemented in the `account_manager` contract so the internal mapping state can be updated to reflect the account ownership properly.

Status: Acknowledged

11. Role-based access control assignment does not follow best practices and centralization issues

Severity: Minor

The `marketplace` contract implements a role-based access control mechanism. It defines two roles: `owner` and `admin`. The roles are set in the constructor function in `contracts/marketplace/lib.rs:210-213`.

However, those roles are both assigned to a single `AccountId`, which is not a valid approach. Role-based access control mechanisms should follow separation of concerns, with each role responsible for a specific aspect of a configuration or certain privileged operations. Assigning multiple roles to the same entity defeats the purpose of implementing a role-based access control scheme.

Similar issues can be observed in `contracts/account_manager/lib.rs:92-95` and `contracts/collection_fabric/lib.rs:116-119`.

Additionally, the privileged roles should only be responsible for maintaining and configuring the protocol itself. If the privileged roles can bypass the business logic to forcefully change the state of the contract, they introduce centralization concerns.

For example, admin roles could bypass the ownership check to execute the `cancel_auction` function in `impls/auction/mod.rs:166-168`, the `start_auction` function in lines `133-135`, and the `cancel_listing` function in `impls/marketplace/mod.rs:95-97`. This may be undesired for the auction/listing creator.

Recommendation

We recommend assigning roles to distinct entities with set responsibilities within the protocol. Alternatively, if business analysis reveals that the granular separation of duties is unnecessary, we recommend simplifying the protocol and removing unnecessary roles.

Furthermore, we recommend that neither role can be used to bypass business logic checks and forcefully change the state of the contract.

Status: Acknowledged

12. Identical token IDs may lead to confusion

Severity: Minor

The `arch_nft` contract implements a `mint_with_metadata` function in `impls/collection/impls.rs:108`, allowing the owner to create new NFTs.

The issue is that the owner needs to provide an ID when creating an NFT. The ID parameter is an enum that contains many variants, such as `U8(u8)`, `U16(u16)`, `U32(u32)`, `U64(u64)`, `U128(u128)`, and `Bytes(Vec<u8>)`. This means the owner can mint `Id::U8(1)` and `Id::U64(1)` as two different NFTs but will be represented as token ID `1` on the front end, potentially confusing users.

Recommendation

We recommend enforcing the usage of a single `Id` variant.

Status: Acknowledged

13. Redundant validations

Severity: Informational

The `buy_nft` function in the `marketplace` contract allows a user to purchase a listed NFT. Several validations are implemented to ensure that only a valid transaction can be completed. In particular, the `assure_transfer` function in `impls/marketplace/mod.rs:148` ensures that the native funds sent are sufficient to cover the cost of the NFT.

However, this validation is redundant because the `transfer_from` function in line `157` also performs the validation in `impls/shared/currency.rs:181`.

Consequently, the same validation is performed twice, unnecessarily increasing the gas consumed for the transaction.

Furthermore, the `bid_nft` function validates the auction's start time to be larger than the current timestamp in `impls/auction/mod.rs:210-212`. This validation is redundant because the `start_auction` function in `impls/auction/mod.rs:141-143` already checks it, which must be called before the bidding phase.

Recommendation

We recommend removing redundant validations.

Status: Acknowledged

14. NFT can be listed as free

Severity: Informational

In `impls/marketplace/mod.rs:42`, the `list_nft_for_sale` function does not validate the `price` parameter, representing the NFT sale price. This means it is possible to list NFTs for free, which may cause creators to lose them if a misconfiguration occurs.

Recommendation

We recommend modifying the `list_nft_for_sale` function to ensure the price of NFT is not zero.

Status: Acknowledged

15. Unnecessary `account_id` function

Severity: Informational

The `arch_nft` contract defines the `account_id` function in `contracts/arch_nft/lib.rs:152-154`, which returns the contract address. This function is not required because the caller needs to know the contract's `AccountId` to call it, defeating the function's purpose.

Recommendation

We recommend removing the `account_id` function.

Status: Acknowledged

16. Code optimizations

Severity: Informational

The `buy_batch` function in `impls/marketplace/mod.rs:178` allows users to purchase multiple NFTs. The function accepts a vector of listing IDs, and for each entry, it ensures a listing is associated with it and validates its state in lines `181-199`. Subsequently, the function parses the vector again to compute the required native funds to be transferred along with the call in lines `201-213`.

Those operations are performed in two separate `fold` executions on two separate iterators, which can be simplified into a single pass over the vector.

Additionally, the `buy_batch` function does not validate the ID vector to contain unique values. If there are duplicate IDs, the whole transaction will be reverted. This could be avoided by filtering unique elements in the vector.

Recommendation

We recommend implementing the optimization stated above.

Status: Acknowledged

17. Missing validation on strings may lead to cross-site scripting

Severity: Informational

In `contracts/arch_nft/lib.rs:123-125`, the `token_name`, `token_uri`, and `additional_info` attributes are not validated or sanitized. If these values are displayed in raw format on the dApp front end, they may be parsed as JavaScript or HTML, opening up client-side attack vectors that may target end-users, such as cross-site scripting (XSS).

This issue is also present in the `collections` creation and `user` object attributes.

Recommendation

We recommend validating the string parameters against an expected format. For example, the values can be restricted to an URI format with special characters are not allowed.

Status: Acknowledged

18. Unused errors

Severity: Informational

In `traits/mod.rs:21`, the `ArchisinalError` enum lists all errors used across the codebase. However, the `CreatorIsNotCaller`, `CallerIsAuctionOwner`, and `AuctionHasNoBids` errors are not used.

Recommendation

We recommend removing the unused errors or implementing them in the codebase.

Status: Acknowledged