AUDIT FOR

ARTZERO

by BRUSHFAM

March 28th, 2023

1. INTRODUCTION

The ArtZero platform aims to be a decentralized NFT marketplace on the AlephZero blockchain. It aims to allow the users to list their NFT collections to be tradeable on the platform for a fee and to create their NFT collection via the ArtZero contracts. The users can create the collections as standard NFT collections or in an advanced mode, which also serves as a launchpad for such projects. The platform also comes with its native NFT Collection, which owners can stake for platform fees and other perks.

Brushfam conducted an Audit, which serves as a technical audit and an implementation audit, focused on **code safety and vulnerability to known issues, poor coding practices, unsafe behavior, leakage of secret or other sensitive data, susceptibility to misuse, error management, error logging, and business logic review.**

2. CONTRACTS INFORMATION

The contracts' code resides in the Azero_Contracts folder, which contains the contracts implementations and the logic of these contracts. We recommend using a more comprehensive project structure, which is addressed in the issue **ARTZ-L01**. The following contracts and their traits were the subjects of this audit:

- **Collection Manager**
- **LaunchPad Manager**
- **Marketplace PSP34**
- **PMP Staking Contract**
- **Standard Launchpad Contract**
- **Standard NFT Contract**

3. AUDIT PROCESS

The audit process consists of three parts:

Initial review

we initially review the project and its contracts and share the audit report internally with the client

Period for applying changes

the client will fix the findings and issues shared in the initial report

Review changes

we review the new updates by the client, along with reviewing other modifications done to the code, and update the review summary accordingly

Brushfam tech team initially identified several issues with the project structure. This was addressed by the client, and since it is not a contract issue, we will not include it in the audit report. All findings were assigned one of the following severity levels according to the importance of the finding. We also assigned status to each finding (R - resolved, A - Acknowledged):

Severity Level	What does it mean	Findings	R	A
Critical	Easily exploitable issues by many actors, or issues that cause high-level code flaws, or can severely harm the users.	1	1	0
High	Issues that can harm the users but are not easily exploitable, or only a specific number of actors can misuse this issue.	4	4	0
Medium	Issues that can harm the users but can be misused only on certain occasions or features that do not work as desired.	3	3	0
Low	Issues that have a low probability of occurring, issues which are not harming users, or slightly affect the performance. Memory footprint issues.	8	7	1
Informational	Code style and logging issues.	10	5	5

FINDINGS

This section will examine the contracts and issues found during the audit.

Severity: Critical

ARTZ-C01 Initialize function can be called by anyone multiple times

Found in **Launchpad Manager**

Initialize function, which initializes the contract storage, can be called by anyone, which **will** lead to a reset of the contract storage, potentially harming the platform users. Only privileged users should be able to call this function, and only if they did not initialize the contract already.

Status: Resolved

Severity: High

ARTZ-H01 Initialize function can be called multiple times

Found in **Collection Manager, Marketplace, Staking**

Initialize function will initialize the contract storage, but the owner can call the function multiple times, leading to a contract reset, which may be an undesired behavior. To avoid this, we recommend checking if the initialize function was called already or making it not a message.

Status: Resolved

ARTZ-H02 Misuse of collection_owner

Found in **Collection Manager**

Functions **auto_new_collection** and **add_new_collection** use the collection owner as a parameter, which malicious actors can misuse. Anyone can create multiple empty NFT collections for an account. While this is unwanted behavior, since the platform may show a famous account as a creator of numerous collections (which also may be malicious), it may make the contract unusable for the specified account since ink! has a size limit to the storage values. The best way to handle this would be to omit the **collection_owner** attribute to this function and work with the **caller** value.

Status: Resolved

Severity: High

ARTZ-H03 Misuse of project_owner

Found in **Launchpad Manager**

Function **add_new_project** uses the project owner as a parameter, which malicious actors can misuse. Anyone can create multiple empty NFT collections for an account. While this is an unwanted behavior since the platform can show a famous account as a creator of numerous collections (which also may be malicious), it may make the contract unusable for the specified account since ink! has a size limit to the storage values. The best way to handle this would be to omit the **project_owner** attribute to this function and work with the **caller** value.

Status: Resolved

ARTZ-H04 Possibility of DOS

Found in **Marketplace**

Function **buy** transfers funds to multiple accounts in a for loop. We expect each transfer to succeed, but if a transfer fails, the contract execution reverts. And since this call can not change the state, the storage will get stuck in the same state. To address this issue, we suggest using a withdraw model, where if a user outbids another user, the contract will save the value that this user can withdraw, and then they can withdraw their bid by calling a **withdraw** function.

Status: Resolved

Severity: Medium

ARTZ-M01 Seller may receive fewer tokens

Found in **Marketplace**

If a seller sells an NFT, they will receive the amount they listed the token for, reduced by the platform fee and the royalty fee. However, these fees are calculated on sale, so a situation like this may occur. Alice lists a token for 1000 tokens, while the platform fee is 2.5% and the royalty fee is 0, expecting 975 tokens. The collection owner sets the royalty fee to 5%, and Bob buys the token from Alice, but Alice only gets 925 tokens.

Status: Resolved

Notes: Information about the royalty fee at the time of listing is saved and used to calculate the fee during the sale.

ARTZ-M02 Unsafe unwrap

Found in **Multiple contracts**

Throughout the contracts, unwrap is called on many places without the assumption of the unwrap returning the **None** variant. This can cause the contract execution to revert with panic, providing no reason. We recommend using the ? operator, using the **unwrap_or_default** function and returning an error on a **None** variant.

Status: Resolved

Severity: Medium

ARTZ-M03 Update collection owner

Found in **Collection Manager**

The function **update_collection_owner** does not remove the collection from the owner's collections, and the removal process can be optimized. The function will **insert** the collection into the old owner's collections instead of removing it. Also, this function finds the collection index within the owner's collections with the time complexity **O(n)**. This can be optimized to **O(1)** by saving the collection index within this user's collections. The removal process of the collection from the user's collection also has a time complexity of **O(n)**. It can be optimized to **O(1)** by swapping the removed collection for the last collection and removing the last collection.

Example 1: Alice has 500 collections and wants to remove the 250th collection. We need to iterate over 250 collections to get the index of the removed collection. Ideally, we can save the index of this collection in the collection data, and we can just retrieve the index within Alice's collections. Note that after implementing this approach, we will also need to update the index of this collection after changing the owner, as two owners will have different collection vectors.

Example 2: Alice has 500 collections and wants to remove the 250th collection. After removing the 250th collection, the vec needs to be shifted by 250 elements. We can avoid this by swapping the removed element (250th in this case) for the last element (500th in this case) and removing the last element, so there will be no need to shift the vec.

Status: Resolved

Severity: Low

ARTZ-L01 Storage is non-upgradeable

Found in **PSP34 Launchpad Standard, PSP34 Standard**

This contract has all the data in the contract struct. One problem with this is that the contract's data could be more transparent when organized into separate structs. Another problem, which would create more problems than the first one mentioned, is that we will not be able to upgrade the storage in the future if we would like to. Also, doing it this way makes the contracts inconsistent since the structs used for contract storage in other contracts.

Status: Acknowledged

Notes: This issue was initially declared as a high-severity issue because of ink! v3 not being stable and redeclared as low-severity, while an NFT smart contract such as this one does not need to be upgradeable. While this smart contract is upgradeable now, it will be complicated to upgrade such a contract since the data struct is declared within the contract. We recommend declaring the data struct and the logic of the contract in a separate crate. However, due to the nature of these smart contracts, they don't need to be upgradeable, and this issue was acknowledged by the ArtZero team.

Severity: Low

ARTZ-L02 Unsafe arithmetic operations

Found in **Multiple contracts**

Several arithmetic operations are performed unsafely, leaving an opportunity for overflow/underflow. These operations should be performed with **checked_add, checked_sub, checked_mul**, and **checked_div**, and check for the result of such operations afterward.

Status: Resolved

Notes: While there may still be unchecked arithmetic operations within the code, we do not find these unsafe due to the nature of what values can end up in such operations.

ARTZ-L03 Constructor execution may revert

Found in **Collection manager**

The contract's owner is initialized with **owner_address**, which comes as the constructor parameter. If the **owner_address** is different from the caller of the constructor, the function **initialize** will revert. Moreover, since we assume the initialize call returns **Ok**, the unwrap call will also fail, reverting without a proper reason. The output of **initialize** can be checked, and the function may panic if the result is **Err**. We also recommend that the contract owner call the constructor and pass **instance.env().caller()** as the parameter of **_init_with_owner**.

Status: Resolved

Severity: Low

ARTZ-L04 Function parameters can use references

Found in **Collection manager**

When passing parameters to a helper function, we recommend passing references (even for primitives), as WASM is more efficient with references. We identified several internal functions which can use references.

Status: Resolved

ARTZ-L05 Attribute keys are not known to the contract

Found in **Collection manager**

The contract does not store the attribute keys, so only the user setting these attributes knows them (if they remember) or a database. So in case of loss of off-chain data, the information about the collection attributes will be lost. Better logic for this would be to have a **Mapping<AccountId, Vec<(String, String)>>**, which maps the account id to a vec of tuples, where the first element would be the attribute key, and the second element would be the value of the attribute.

Status: Resolved

Severity: Low

ARTZ-L06 Token owner can not call lock function

Found in **PSP34 Launchpad Standard**

According to the logic, an owner of a token should be able to call the **lock** function. But the contract checks if the caller is the owner of the contract.

Status: Resolved

ARTZ-L07 Inoptimal key type

Found in **PSP34 Launchpad Standard, PSP34 Standard**

The type of the storage field attribute_names may be changed to **Mapping<String, ()>** to optimize the **add_attribute** function.

Status: Resolved

Notes: This issue is bound to the **ARTZ-L08** issue.

Severity: Low

ARTZ-L08 Attribute name logic can be optimized

Found in **PSP34 Standard**

The time complexity of adding and retrieving attributes and attribute names can be optimized by adding another **Mapping** or using **MultiMapping**. The functions **add_attribute_name** and **get_attribute** have a time complexity of **O(n)**, with **n** being the number of currently existing attributes. This can be optimized to **O(1)** time complexity by mapping the attribute name to its value and the index of this attribute to an empty value (note that this value can be anything, but because of the nature of this field, an empty value or a boolean value would be most suitable). You can achieve this by introducing a **Mapping<u32, String>** to keep track of attribute names available and a **Mapping<String,()>** to keep track of existing attribute names.

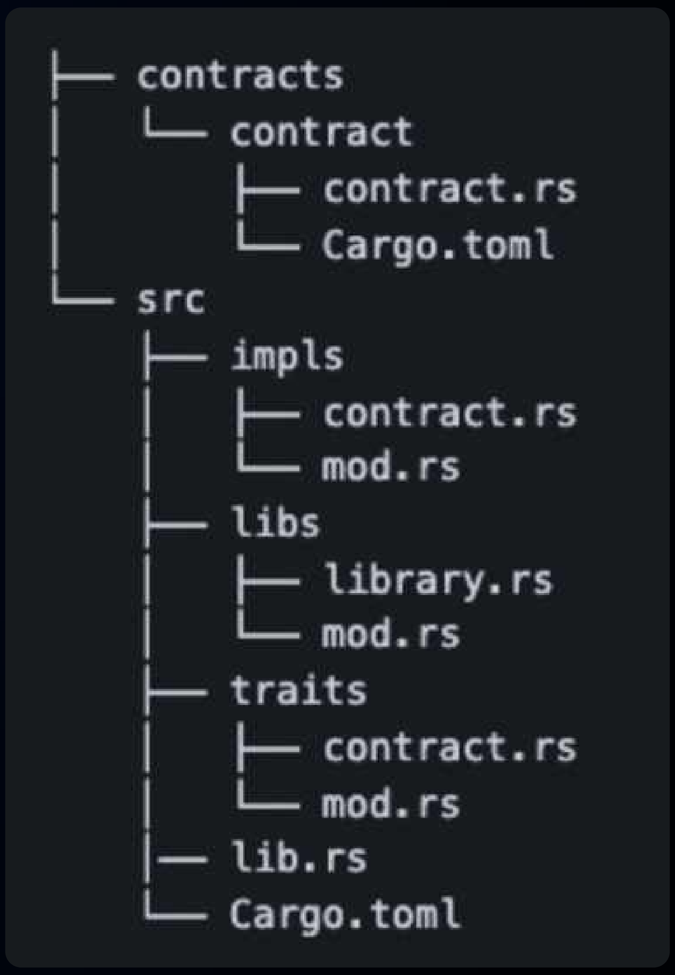
Status: Resolved

Severity: Informational

ARTZ-101 Project structure

Found in **Multiple contracts**

Rust and ink! allow the developers to write the smart contracts in a more architecture-friendly way. We encourage this, as it makes the projects more readable and easily maintainable. We recommend breaking the structure of the contracts into the following structure:



```
├── contracts
│   └── contract
│       ├── contract.rs
│       └── Cargo.toml
└── src
    ├── impls
    │   ├── contract.rs
    │   └── mod.rs
    ├── libs
    │   ├── library.rs
    │   └── mod.rs
    ├── traits
    │   ├── contract.rs
    │   └── mod.rs
    |── lib.rs
    └── Cargo.toml
```

In this structure, the logic of the contracts would be implemented within the src crate, and the ready-to-deploy contracts would reside in the contracts folder, with every contract being a separate crate.

Status: Acknowledged

Severity: Informational

ARTZ-I02 Missing events for significant storage changes

Found in **Multiple contracts**

It is a good practice to emit an event on significant storage changes since there might be other services or APIs listening to such events, making the application more accessible and transparent. We outlined several functions to the ArtZero team where an event might be emitted.

Status: Acknowledged

ARTZ-I03 Possibility to use an enum

Found in **Collection manager**

The field **contract_type** in the **Collection** structure can be of type **enum ContractType**, instead of an u8. Using an enum will make understanding the field's value in the code easier, increasing readability and easing maintenance.

Status: Resolved

Severity: Informational

ARTZ-I04 Possibility to use modifiers

Found in **Multiple contracts**

In many places throughout the contracts, there are checks for the same condition. This is duplicity and can be avoided by a custom modifier or a helper function.

Status: Acknowledged

Notes: We recommend checking the contracts for duplicity and removing them.

ARTZ-I05 Insufficient rights

Found in **Multiple contracts**

Only an admin can call some functions, but not the owner. Since the owner is a higher role than the admin, the owner should also have the admin's rights. We recommend the usage of an admin role or the use of the AccessControl trait to manage roles.

Status: Acknowledged

Notes: The access roles of the contracts were improved. However, there is still an unclear hierarchy of the roles.

Severity: Informational

ARTZ-106 Unnecessary Option

Found in **PSP34 Launchpad Standard**

Functions **get_phase_schedule_by_id** and **get_whitelist_by_account_id** return Option<T>, but the values returned originate from a Mapping. The return statement can be simplified to **self.phases.get(&phase_id)** and **self.phase_whitelist_link.get(&(account, phase_id))**.

Status: Resolved

ARTZ-107 Inoptimal type used

Found in **PSP34 Standard**

The variable **locked_tokens** is only used to check if the tokens are locked; therefore, we recommend using bool or enum type.

Status: Resolved

Severity: Informational

ARTZ-108 Inconsistent error handling

Found in **Marketplace**

The function **receive_hold_amount** uses assert macro to handle the transfer. We recommend sticking to the same error handling throughout the project, meaning using Result here.

Status: Resolved

ARTZ-109 Unnecessary unwrap

Found in **Multiple contracts**

The functions **get_for_sale_token_id (Marketplace)**, **get_project_by_nft_address (Launchpad Manager)**, and **get_collection_owner (Collection Manager)** return an Option. However, the value of this Option is the result of another unwrap. This can be simplified without using **Some()**, **unwrap()**, and also the functions from Option might be helpful.

Status: Resolved

Severity: Informational

ARTZ-I10 Reentrancy allowed on PSP22 and PSP34 transfer

Found in **AdminTrait**

The functions **tranfer_psp22** and **tranfer_nft** transfer PSP22 and PSP34 tokens, respectively, with reentrancy allowed. These functions are used to withdraw tokens sent to the contract by mistake. The ArtZero team should exercise caution while using this function, as the code of the PSP22 or PSP34 might reenter the contract and potentially cause harm.

Status: Acknowledged

CONCLUSION

Brushfam tech team analyzed the smart contracts within the scope according to the technical documentation provided during the initial review. We manually reviewed these smart contracts and analyzed the rust code with the available tools. We focused on the code structure, code cleanness, ink! and OpenBrush specifics, and vulnerability to known issues. During the initial review, we identified several general issues regarding the project's architecture, which were discussed with the ArtZero team and addressed.

We identified one critical severity issue, four high severity issues, three medium severity issues, eight low severity issues, and ten informational issues. All of these issues were resolved or acknowledged by the ArtZero team. We provided an audit report to the ArtZero team containing all the findings.

DISCLAIMER

As of the date of this audit, the audit of ink! has yet to be finished, meaning potential vulnerabilities within its syntax and implementation may occur. Therefore, unknown vulnerabilities originating from the ink! language may present themselves and go undetected by this audit process.

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we did our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

CONTACTS

BD — Alina Antropova

@alantropova

alina@727.ventures

brushfam.io

BRUSHFAM