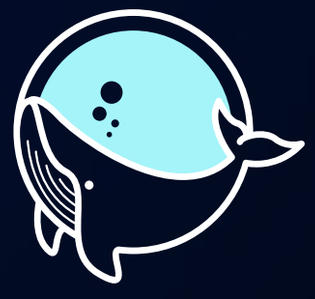
AUDIT FOR



Ink Whale

by BRUSHFAM

October 13th, 2023

1. INTRODUCTION

The InkWhale platform aims to be a smart contract token launchpad on the AlephZero blockchain. It aims to allow the users to create a launchpad for their PSP22 token, with which other users can invest in the token via a public sale or a whitelist sale. A launchpad can be created and added to the platform for a fee paid in INW token.

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

2. CONTRACTS INFORMATION

The application consists of several contracts implemented in the InkWhale GitHub repository. The following contracts and their traits were the subjects of this audit:

1. Launchpad Generator

This contract is the main entry point for launchpad creation. Users interact with this contract to create a launchpad for their token, which is then registered to the platform.

2. Launchpad Contract

This contract is the core of the platform, as users can interact with specific launchpad contracts to invest in a public sale or a whitelist sale of a specific token.

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 has worked with the client on a synchronized basis, meaning while we were conducting the audit, we notified the client about new findings every time some findings occurred. The client then had the time to fix the found issues.

We initially identified several issues with the contracts that were the subject of this audit. All findings were assigned one of the following severity levels according to the importance of the finding. All of the issues reported to the client were resolved during the audit process.

Severity Level	What does it mean	Findings	R
Critical	Easily exploitable issues by many actors or issues that cause high-level code flaws or can severely harm the users.	1	1
High	Issues that can harm the users but are not easily exploitable, or only a specific number of actors can misuse this issue.	1	1
Medium	Issues that can harm the users but can be misused only on certain occasions or features that do not work as desired.	0	0
Low	Issues with a low probability of occurring, issues that do not harm users, or slightly affect the performance. Memory footprint issues.	5	5
Informational	Code style and logging issues	11	11

FINDINGS

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

Severity: Critical

INKW-C01 Malicious logic can be injected to launchpad contract

Found in **LaunchpadContract**

The launchpad contract is instantiated every time a new launchpad is created and only lasts while the launch phase is running. So, by design, such a contract doesn't need to be upgradeable, as the only thing that we need to change for the protocol to start working with a new launchpad contract version is to change the code hash that the launchpad contracts will be instantiated with.

However, in this instance, the project creators can misuse the upgradeability to harm their users. Since the upgradeability is restricted to **only_owner**, and the owner role is granted to the launchpad owner, they are able to upgrade the code of their launchpad to a malicious one, which could potentially harm users.

Consider removing the upgradeability from the Launchpad contract.

Status: Resolved

Severity: High

INKW-H01 Surplus native won't be returned

Found in **LaunchpadContractTrait**

Functions **public_purchase** and **whitelist_purchase** do not account for the possibility that a user might send more native tokens than needed for the purchase. Suppose Alice wants to purchase 1000 tokens sold at 1 AZERO per token. Alice sends this transaction with 1500 AZERO as value and will get her 1000 tokens, but the price for these tokens is 1000 AZERO, and the remaining 500 AZERO will be stuck in the contract.

Consider adding a check, which ensures that when a user sends more native tokens than needed for the purchase, the surplus native tokens will be returned to the caller.

Status: Resolved

Severity: Low

INKW-L01 Tokens are transferred twice

Found in **LaunchpadGenerator**

When calling the **new_launchpad** function, tokens meant for the launchpad are sent from the caller to the contract and then from the contract to the newly created launchpad contract via the topup function. This can be optimized so that the tokens are sent directly from the caller to the new contract, which would omit the calling of one transfer function and the approve function.

Consider calling **Psp22Ref::transfer_from(&token_address, caller, contract_account, total_supply, Vec::<u8>::new())** once the new launchpad contract is instantiated and removing the **PSP22Ref::approve** call, instead of **Psp22Ref::transfer_from(&token_address, caller, self.env().account_id(), total_supply, Vec::<u8>::new())**.

Status: Resolved

Severity: Low

INKW-L02 Needless contract message

Found in **LaunchpadContract**

Function **initialize** will set up the default admin role and the **ADMINER** role for the method caller (the contract owner) and then call the **create_pool** function. This call will fail if the **generator_contract** is not set to **0x00**. Since it only makes sense for users to deploy this contract with the LaunchpadGenerator contract (otherwise, it would not be registered to the platform), which also sets the **generator_contract** to the correct value, calling this method will always fail, therefore, it is useless.

Consider removing the needless **initialize** function.

Status: Resolved

Severity: Low

INKW-L03 Code repetition

Found in **LaunchpadContractTrait**

The logic for claiming tokens in functions public_claim and whitelist_claim only differs in the data to be updated. This functionality can be done in one function to avoid duplicate code.

Consider creating a function that would take the respective information and return the data that should be updated on the blockchain.

Status: Resolved

INKW-L04 Adding a new phase without vesting fails

Found in **LaunchpadContract**

Users who want to add a new phase without vesting will intuitively call the **add_phase** function with **vesting_unit** and **vesting_duration** both set to 0. However, the call will fail since the **vesting_unit** must not be zero.

The same problem is present in the **set_vesting_unit** function in LaunchpadContractTrait.

Status: Resolved

Severity: Low

INKW-L05 Allowed reentrancy on PSP-22 transfers

Found in **Multiple contracts**

Several methods transfer PSP-22 tokens with the reentrancy flag set to true. While this was required in the previous versions of OpenBrush, the reentrant call **do_safe_transfer_check** was removed from PSP-22 in OpenBrush version 3.1.0. Therefore, there is no more need for this flag to be enabled. While we have not found any vulnerabilities connected to the reentrant calls, we recommend removing these as they are not needed anymore.

Status: Resolved

Severity: Informational

INKW-I01 Incompatibility with the newest cargo-contract

Found in **Multiple contracts**

Using the latest development tools and library releases is the best practice. The cargo contract crate had a recent update to version 3.2.0, and we recommend using this version of the tool, which also requires small updates in the contract file.

Status: Resolved

INKW-I02 Redundantly disabled clippy lints

Found in **Multiple contracts**

Some clippy lints were disabled in contracts, however, disabling them has no effect, as removing these will not produce any warnings on **cargo clippy**. The following lints can be removed: **inline_fn_without_body**, **large_enum_variant**.

Status: Resolved

Severity: Informational

INKW-I03 Handling of functions that return errors

Found in **Multiple contracts**

When handling a function that may return an error, a variable with the result is initialized. This is unnecessary since the result is only used to return the error if it is an **Err** variant. The ? operator can be used instead.

Status: Resolved

INKW-I04 Unneeded return statement

Found in **LaunchpadContract**

Removing the return and semicolon will make the code more rusty.

Status: Resolved

Severity: Informational

INKW-I05 Unnecessary clone

Found in **LaunchpadContract**

In some places, the **clone** function is called even though the type implements copy.

Status: Resolved

INKW-I06 Handling of Option reading

Found in **LaunchpadContract**

In multiple places, you read a field of an optional field and return **Some** if the field is **Some** or **None** otherwise.

Status: Resolved

Severity: Informational

INKW-I07 Parameters can be bundled into a struct

Found in **LaunchpadContract**

Functions **new**, **initialize**, and **create_pool** all take 9 parameters regarding phase. This can be bundled into one **Vec<PhaseInfo>.** This will make the code more readable and allow you to omit some checks for correct vec length. The same thing goes for functions **set_multi_phases** and **set_phase** in the **LaunchpadContractTrait**.

Status: Resolved

INKW-I08 Useless data structs

Found in **Multiple contracts**

The **Data** struct only contains one field **_reserved: Option<()>**, which is not used. Upgrade of storage can be done without this approach since ink! v4, therefore, this struct can be deleted, along with the need for the **UpgradeableTrait** to implement **Storage<Data>**. The same goes for the **Data** struct in admin. These traits depend only on the **ownable::Data** struct, so there is no need for the empty **Data** structs and the implementation of **Storage<Data>** for the respective structs.

If the **AdminTrait** or the **UpgradeableTrait** will need some new storage fields, it is possible to add the Data struct and the implementation with the required fields. However, having the **_reserved: Option<()>** field is no longer needed, as new fields can be added to an existing data struct.

Status: Resolved

Severity: Informational

INKW-I09 Useless STORAGE_KEY constants

Found in **Multiple contracts**

The **STORAGE_KEY** is useless in OpenBrush v4. It was initially meant to make the contracts upgradeable with the specific storage keys, but OpenBrush currently handles this in the **storage_item** macro. These storage keys are also used in other traits to declare a derive of **StorableHint** and **AutoStorableHint**, which is also unnecessary.

Status: Resolved

INKW-I10 Getter for balance with access restriction

Found in **Multiple contracts**

Method **get_balance** returns the contract balance and has the **only_owner** modifier. Anybody can query the balance of an account without a contract message, which would bypass this restriction. Therefore, this restriction is pointless. Moreover, it requires the method to be mutable, while it does not mutate the contract storage and should be declared as **&self** instead of **&mut self**.

Status: Resolved

Severity: Informational

INKW-I11 Needless message

Found in **LaunchpadGeneratorContract**

Method **initialize** can only be called once, and it is called during the instantiation of the contract, making it uncallable after the deployment of the contract, but making it a message hints that it can be called multiple times.

Consider making this function an internal function of the contract.

Status: Resolved

CONCLUSION

We analyzed the smart contracts within the scope according to the inline documentation, information provided during the initial review, and information collected from the client during the audit process. 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, vulnerability to known issues, and specifics of launchpad platforms.

We identified one critical severity issue, one high severity issue, five low severity issues, and eleven informational issues. We provided an audit report to the InkWhale team containing all the findings.

With our guidance, the InkWhale team has resolved all of the issues found during the audit process.

DISCLAIMER

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