Audit Report

Ninja

v1.0

January 20, 2023

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

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

Introduction

Purpose of This Report

Oak Security has been engaged by Injective Labs to perform a security audit of the Ninja Finance vaults.

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 GitHub repository:

https://github.com/ninja-finance/ninja-contracts

Commit hash: `656666d073cc2108f9dd566e9a3fb6e394a4440f`

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

Ninja Finance is a smart contract application built on top of the Injective blockchain leveraging its exchange and autonomous execution capability. Ninja Finance enables users to interact with both spot and derivative vaults that are automatically managed each block by an on-chain bot.

The audit scope comprehends the master contract, which is responsible for managing all the protocol vaults, and an example implementation of spot and derivative vaults.

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.

Summary of Findings

No	Description	Severity	Status
1	Fees held by spot vault are locked forever	Critical	Resolved
2	Inventory imbalance under certain parameter constellations will disable the risk management framework	Major	Acknowledged
3	Price threshold for price invalidity is prone to manipulation and might lead to full loss of users funds in case of a breakdown of a market or an asset	Major	Acknowledged
4	Updating Ninja token address may cause state conflicts	Minor	Resolved
5	Vault contract invocations should be prevented prior to registration with master contract	Minor	Acknowledged
6	Large `order_density` could lead to the execution running out of gas	Minor	Resolved
7	Insufficient validation of addresses in configs	Minor	Resolved
8	A dependency is vulnerable to two `CVEs`	Minor	Resolved
9	Custom calls between Cosmos SDK modules and CosmWasm contracts	Minor	Resolved
10	Vault contracts ownership cannot be transferred	Minor	Resolved
11	Message attributes are defined and stored with type inconsistency	Minor	Resolved
12	Parameter space of derivative vault config is loosely validated	Minor	Resolved
13	Backward looking volatility can lead to inefficient pricing	Minor	Acknowledged
14	Math utils do not follow best practices	Minor	Acknowledged
15	Query does not implement a pagination mechanism	Minor	Resolved
16	Use of magic numbers	Informational	Resolved

17	Outstanding `TODO` and `FIXME` comments present in the codebase	Informational	Resolved
18	Redundant validation of variables	Informational	Acknowledged
19	Overflow checks not enabled for release profile	Informational	Acknowledged
20	Contracts should implement a two step ownership transfer	Informational	Acknowledged
21	Custom access control implementation	Informational	Acknowledged

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	Rust language best practices are not always followed. E.g. the use of clone and struct declarations.
Code readability and clarity	Low-Medium	Lack of comments for complex functionality negatively impacts the readability and clarity of the codebase.
Level of documentation	Low-Medium	The documentation has some inconsistencies and varies in quality and completeness between different contracts.
Test coverage	High	`cargo-tarpaulin` reports a 93.99% code coverage.

Detailed Findings

1. Fees held by spot vault are locked forever

Severity: Critical

During the instantiation of a spot vault contract the `fee_recipient` is defined as the contract itself, see `contracts/ninja-vault-spot/src/contract.rs:31`. However, there is no function that allows the fees accrued to be withdrawn, effectively locking the funds forever.

Recommendation

We recommend implementing a mechanism that enables the redistribution of fees accrued by the spot vault or otherwise following the pattern of the derivative vault whereby fees are sent to the master contract.

Status: Resolved

2. Inventory imbalance under certain parameter constellations will disable the risk management framework

Severity: Major

In order to conduct risk management, the team modeled a reservation price that is different from the current mid-market price to be the center of the orders. To account for changes in volatilities, the orders price is a function of the volatility. However, for every imbalance ratio there exists a parameter combination of `reservation_price_sensitivity_ratio` and `reservation_spread_sensitivity_ratio` such that the proposed head and tail can be set to market price, by supplying liquidity or withdrawing liquidity, irrespective of the state of volatility.

Specifically in `contracts/ninja-vault-spot/src/mm_bot/order_management.rs:17`, the variable `proposed_buy_head` can be equal to

$$mid\_price + volatility \cdot \beta$$

where

$$\beta = inventory\_imbalance\_ratio \cdot reservation\_price\_sensitivity\_ratio - \frac{reservation\_spread\_sensitivity\_ratio}{2}$$

can be zero if:

$$reservation\_spread\_sensitivity\_ratio \leq 2 \cdot reservation\_price\_sensitivity\_ratio$$

An attacker with access to significant liquidity could leverage this property during market turmoil (i.e. states of high volatility and trending price behavior) to disable the risk management framework, as described above, pushing the vault to an imbalance such that $\beta = 0$.

Recommendation

We recommend implementing a risk management logic such that both volatility and imbalance are taken into account separately and irrespective of the state of the other. No imbalance ratio should allow to cancel the sensitivity to volatility and no volatility state should allow to cancel the sensitivity to the vault's imbalance.

Status: Acknowledged

3. Price threshold for price invalidity is prone to manipulation and might lead to full loss of users funds in case of a breakdown of a market or an asset

Severity: Major

In `contracts/ninja-vault-derivatives/src/mm_bot/bot.rs:40` a 30% interval is used to render the mark price as invalid. If the new price is out of this interval, the logic in `contracts/ninja-vault-derivatives/src/mm_bot/risk_management/oracle.rs` closes positions if profitable, or leaves them open and stops all trading activities if not profitable.

Although this is meant to be a protection against erroneous or manipulated prices, its efficacy is low as an attacker that is capable of manipulating the price by 30%, will also be able to manipulate the price by 29% in one block and 2% in the next or simply manipulate the price by 29.99%. In case of a non-manipulated change by 30% and a continuing trend in the same direction, a losing position will never close – leading to a full loss of funds for users.

There is a way to recover if an admin updates the `last_valid_mark_price` in in `contracts/ninja-vault-derivatives/src/config.rs:91`. However, this is an inadequate risk management tool for an automated trading system with multiple vaults running simultaneously 24/7. In addition, it goes against best practices that admins can update prices as enabled.

Recommendation

We recommend revising the risk management framework to work properly in strong market movements. This should include stop-loss logic, logic that discovers trending price behavior, logic that determines the thresholds more data-based (e.g. on the basis of confidence intervals), and logic that uses volume-weighted price movements to determine the validity of prices.

In addition we recommend disabling the ability to update the price in the config. If desired an admin-permissioned function that closes the current position at all costs could be implemented.

Status: Acknowledged

4. Updating Ninja token address may cause state conflicts

Severity: Minor

The `owner` of the master contract has the ability to update the address of the Ninja token stored in the contract's config. Altering the Ninja token address could have the effect of causing a state conflict in other contracts that use the address stored within the master contract.

Recommendation

We recommend that config parameters that are used by external smart contracts are only updated using migration, as opposed to simple config updates, so that potential state conflicts can be handled during migration.

Status: Resolved

5. Vault contract invocations should be prevented prior to registration with master contract

Severity: Minor

Vault contracts are designed to be deployed and instantiated from their `owner` and then successfully initialized from the `ninja-master` executing the `RegisterForMaster` transaction.

This message execution sets the `subaccount_id` and instantiates the `CW20` contract responsible for managing the vault's `lp_tokens`.

Before this initialization, the vault is not ready to be used and user space transactions and queries should return a meaningful error message.

Recommendation

We recommend implementing a guard in transaction and query handlers in order to ensure that `subaccount_id` and `lp_token_address` are set, and returning an error otherwise.

Status: Acknowledged

6. Large `order_density` could lead to the execution running out of gas

Severity: Minor

In `contracts/ninja-master/src/contract.rs:111-136`, the following functions

- `get_is_authorized_order_data`,
- `get_is_authorized_spot_order`, and
- `get_is_authorized_derivative_order`

perform an unbounded iteration through orders.

Since the length of the vector is related to `order_density` and this parameter has not an upper bound, a large value assigned to it could lead the execution to run out of gas.

An analogous problem also occurs in `contracts/ninja-master/src/contract.rs:336-375` with `synthetic_trade.user_trades`.

Recommendation

We recommend implementing an upper bound for `order_density` parameter to prevent out-of-gas errors.

Status: Resolved

7. Insufficient validation of addresses in configs

Severity: Minor

In the spot vault config validation, a number of fields are validated to ensure they are not empty, i.e. `uninitialised`. However, the addresses may nonetheless be invalid as no additional validation is performed for the following fields:

- `contracts/ninja-vault-spot/src/config.rs:124-126`
- `contracts/ninja-vault-spot/src/config.rs:210-212`

This may cause any queries using these variables to fail and prevent functions from executing.

Recommendation

We recommend providing additional validation of the values mentioned above to ensure they are valid addresses.

Status: Resolved

8. A dependency is vulnerable to two CVEs

Severity: Minor

The `chrono` crate defined as a `dependency` in lines:

- `contracts/ninja-vault-spot/Cargo.toml:28`
- `contracts/ninja-vault-derivatives/Cargo.toml:27`

is vulnerable to two `CVE`s:

- RUSTSEC-2020-0159: chrono
- RUSTSEC-2020-0071 - Potential segfault in the time crate

Recommendation

We recommend updating the `chrono` crate to the version suggested in `CVE` descriptions.

Status: Resolved

9. Custom calls between Cosmos SDK modules and CosmWasm contracts

Severity: Minor

Cosmos SDK modules are performing authorized calls to particular CosmWasm contract handlers using custom logic.

In order to do so, Cosmos SDK modules are calling `Execute` functions from `wasm` module impersonating the receiver contract. From the CosmWasm side, it is as if the message comes from the contract itself.

This means that on the CosmWasm contract, developers should implement custom logic to authorize the call. This has the side effect that the contract can execute Cosmos SDK reserved calls by itself.

Also, since `Execute` in this case is using the same address for the sender and receiver, if the transaction includes some `funds`, the module needs to perform it in two steps. The first one will transfer funds from the module to the `ninja-master` contract and then the second one will call the `wasm Execute` function that will perform the required transaction and, as a side effect, a fake funds transfer from sender to sender.

Recommendation

Since CosmWasm 1.0 there is the concept of `Sudo` messages that are battle tested and designed for this specific use case.

We recommend using such `Sudo` messages to perform calls from Cosmos SDK modules to CosmWasm smart contracts.

Status: Resolved

10. Vault contracts ownership cannot be transferred

Severity: Minor

The `ninja-vault-spot` and `ninja-vault-derivatives` contracts do not implement an ownership transfer mechanism.

Since the defined `owner` account has the right to modify critical vault parameters, a compromise of the owner would have devastating consequences for the vaults.

It is best practice to offer functionality to transfer the ownership to another account.

Recommendation

We recommend implementing a mechanism to transfer the contract ownership to another account following the practices recommended in the issues "Contracts should implement a two step ownership transfer" and "Custom access control implementation" below.

Status: Resolved

11. Message attributes are defined and stored with type inconsistency

Severity: Minor

Contract messages define and store all attributes as a `String` including when another type is more appropriate.

Consequently, during execution variables must be cast and validated which causes inefficiencies. Also, having messages with attributes correctly defined can help users to better understand which data they can provide.

Recommendation

We recommend reworking message definitions in order to improve their structure and adopt types that better fit represented data.

Status: Resolved

12. Parameter space of derivative vault config is loosely validated

Severity: Minor

In `contracts/ninja-vault-derivatives/src/config.rs:35-92` and `contracts/ninja-vault-derivatives/src/contract.rs:51-68`, the parameters are loosely validated with `parse_dec` and `parse_int` functions from `contracts/ninja-vault-derivatives/src/utils.rs:79-96`. Both `parse_dec` and `parse_int` do not check for equality of the parsed parameters to the supposed minima and maxima, which is inconsistent with the validation that is used in the spot vault. In addition, some parameters do not have a minimum or maximum. For example `leverage` doesn't have an upper bound, which can lead to admin or config errors (assuming that e.g. a value of 1,000,000 would be not intended) .

Recommendation

We recommend revising validation of all parameters. From the code and the documentation it is not possible to conduct a risk analysis of the whole parameter space or infer which values are intended.

Status: Resolved

13. Backward looking volatility can lead to inefficient pricing

Severity: Minor

The volatility used in the risk management is backward looking - i.e. a function of realized prices. Empirically, it has been documented that volatility changes also in deterministic patterns (see for example volatility smiles in traditional finance). If the data generating process of volatility contains such deterministic (e.g. hourly, weekly or volume related) patterns, backward looking symmetrically sampled volatility is inefficient for risk management, such that elaborate traders can trade against it with a positive expected value, and in turn an expected loss for users.

Recommendation

We recommend implementing a model-based volatility that accounts for deterministic patterns. This could be for example implemented with bayesian techniques or or simple seasonal or hourly adjustment parameters.

Status: Acknowledged

14. Math utils do not follow best practices

Severity: Minor

In `contracts/ninja-vault-spot/src/utils.rs:10` and `18`, division by zero returns zero. Similarly, in `contracts/ninja-vault-spot/src/utils.rs:34`, `42`, and `50`, subtractions that would result in values smaller than zero return always zero. It is best practice to return errors or panic in such cases, as any changes to that standard behavior of math functions may confuse developers and may introduce bugs in the future.

Recommendation

We recommend following the best practices errors or panicing in the cases outlined above. Should code need to continue following the return of an appropriate error this should be handled explicitly in the code logic.

Status: Acknowledged

15. Query does not implement a pagination mechanism

Severity: Minor

The `get_registered_vaults` query handler in `contracts/ninja-master/src/queries.rs:40-64` does not implement a pagination mechanism.

This could lead to a computational node overload or, in case the query is executed in a transaction, to an out-of-gas error.

Recommendation

We recommend implementing a pagination mechanism in the mentioned query.

Status: Resolved

16. Use of magic numbers

Severity: Informational

Throughout the code base so-called “magic numbers” are used, i.e. hard-coded numbers without context or other description. Using magic numbers goes against best practices as they reduce code readability and maintenance as developers are unable to easily understand their use and may make inconsistent changes across the code base.

Instances of magic numbers are listed below:

- `contracts/ninja-vault-spot/src/begin_blocker.rs:25`

- `contracts/ninja-vault-derivatives/src/begin_blocker.rs:55`
- `contracts/ninja-vault-derivatives/src/begin_blocker.rs:12`
- `contracts/ninja-vault-derivatives/src/mm_bot/risk_management/volatility.rs:38`
- `contracts/ninja-vault-spot/src/mm_bot/bot_utils.rs:62`
- `contracts/ninja-vault-spot/src/mm_bot/risk_management/volatility.rs:28`

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Resolved

17. Outstanding `TODO` and `FIXME` comments present in the codebase

Severity: Informational

`TODO` and `FIXME` comments were found in the following locations:

- `contracts/ninja-vault-spot/src/config.rs:123`
- `contracts/ninja-vault-spot/src/subscriptions.rs:88`
- `contracts/ninja-master/src/queries.rs:35`, `80`, and `104`
- `contracts/ninja-master/src/state.rs:19`

Recommendation

We recommend resolving the `TODO` and `FIXME` comments in the codebase.

Status: Resolved

18. Redundant validation of variables

Severity: Informational

The variables `total_quote_funds_supplied_amount` and `total_base_funds_supplied_amount` are validated to ensure they are not less than zero in `contract/ninja-vault-spot/src/subscription.rs:94-100`. It is impossible for this scenario to occur as previously the variables are cast from a variable type `u128` in `contract/ninja-vault-spot/src/subscription.rs:55-56`. Performing this check adds complexity to the codebase and increases the computation resources required.

Additionally, when a user subscribes to a derivative vault the `margin_ratio` is validated to ensure it is not less or equal to zero in `contracts/ninja-master/src/contract.rs:507-511`. However, this check is subsequently replicated in `contracts/ninja-vault-derivatives/src/subscriptions.rs:36-38`. This means that the second checks are unreachable and increase code complexity.

Recommendation

We recommend removing the redundant checks in `contract/ninja-vault-spot/src/subscription.rs:94-100` and performing the validations of `margin_ratio` once in `contracts/ninja-vault-derivatives/src/subscriptions.rs` and eliminating the duplicated validation.

Status: Acknowledged

19. Overflow checks not enabled for release profile

Severity: Informational

The following packages and contracts do not enable `overflow-checks` for the release profile:

- `contracts/ninja-master/Cargo.toml`
- `contracts/ninja-vault-spot/Cargo.toml`
- `contracts/ninja-vault-derivatives/Cargo.toml`

While enabled implicitly through the workspace manifest, a future refactoring might break this assumption.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Acknowledged

20. Contracts should implement a two step ownership transfer

Severity: Informational

The `ninja-master` contract allows the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to the incorrect address. A two-step ownership

transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated and lowercased.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

21. Custom access control implementation

Severity: Informational

Protocol contracts implement custom access controls. Although no instances of broken controls or bypasses have been found, using a battle-tested implementation reduces potential risks and the complexity of the codebase.

Also, the access control logic is duplicated across the handlers of each function, which negatively impacts the code's readability and maintainability, as it is error-prone.

Recommendation

We recommend using a well-known access control implementation such as `cw_controllers::Admin` (https://docs.rs/cw-controllers/0.14.0/cw_controllers/struct.Admin.html).

Status: Acknowledged