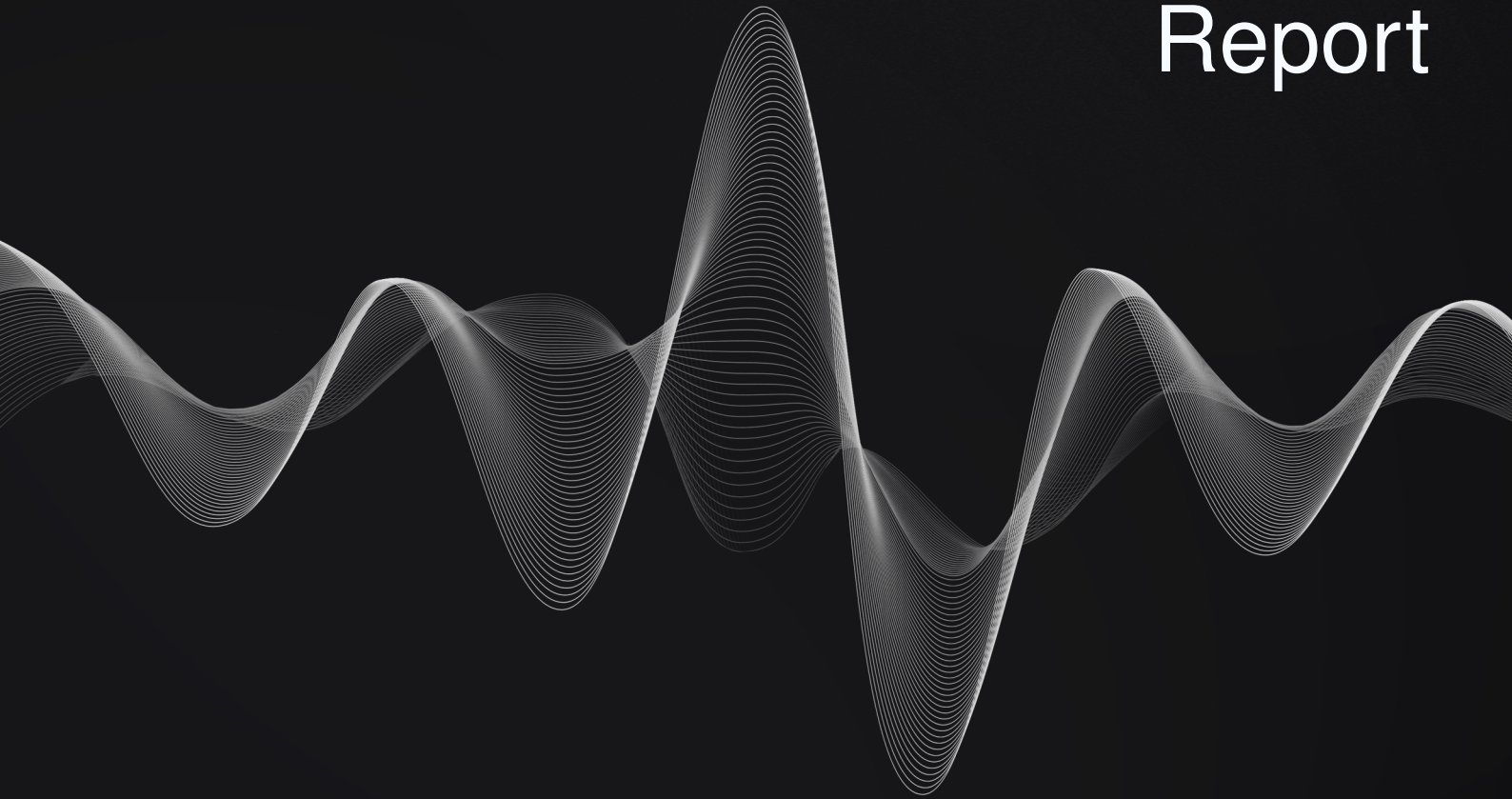




# Andromeda

## Andromeda CosmWasm Audit Report










# Document Control

**CONFIDENTIAL**

**FINAL**(v2.0)

## Audit\_Report\_ANDR-PRO\_FINAL\_20

Nov 10, 2023		v0.1	Michał Bazyli: Initial draft
Nov 15, 2023		v0.2	Michał Bazyli: Added findings
Nov 16, 2023		v0.3	Michał Bazyli: Draft
Nov 16, 2023		v1.0	Charles Dray: Approved
Nov 20, 2023		v1.1	Michał Bazyli: Added findings
Mar 19, 2024		v1.2	Michał Bazyli: Reviewed findings
Mar 19, 2023		v2.0	Charles Dray: Finalized

### Points of Contact

Mike Grantis  
Charles Dray  
Luis Lubeck

Andromeda  
Resonance  
Resonance

mike.grantis@gmail.com  
charles@resonance.security  
luis@resonance.security

### Testing Team

Michał Bazyli  
Ilan Abitbol  
João Simões

Resonance  
Resonance  
Resonance

michal.bazyli@resonance.security  
ilan.abitbol@resonance.security  
joao.simoes@resonance.security

## Copyright and Disclaimer

© 2024 Resonance Security, Inc. All rights reserved.

The information in this report is considered confidential and proprietary by Resonance and is licensed to the recipient solely under the terms of the project statement of work. Reproduction or distribution, in whole or in part, is strictly prohibited without the express written permission of Resonance.

All activities performed by Resonance in connection with this project were carried out in accordance with the project statement of work and agreed-upon project plan. It's important to note that security assessments are time-limited and may depend on information provided by the client, its affiliates, or partners. As such, the findings documented in this report should not be considered a comprehensive list of all security issues, flaws, or defects in the target system or codebase.

Furthermore, it is hereby assumed that all of the risks in electing not to remedy the security issues identified henceforth are sole responsibility of the respective client. The acknowledgement and understanding of the risks which may arise due to failure to remedy the described security issues, waives and releases any claims against Resonance, now known or hereafter known, on account of damage or financial loss.

# Contents

<b>1 Document Control</b>	<b>2</b>
Copyright and Disclaimer .....	2
<b>2 Executive Summary</b>	<b>4</b>
System Overview .....	4
Repository Coverage and Quality.....	4
<b>3 Target</b>	<b>6</b>
<b>4 Methodology</b>	<b>7</b>
Severity Rating.....	8
Repository Coverage and Quality Rating.....	9
<b>5 Findings</b>	<b>10</b>
Lack Of Module Registration Enforcement Mechanism In Contracts .....	12
Inefficient Validation In Modules Registration Process .....	13
Incorrect Calculation Of Asset To Send In Sale Ratio Calculation .....	14
The Unfair Advantage Of The First Staker .....	15
Lockdrop Participants Won't Be Able To Withdraw Native Assets .....	16
origin Of The Packet Can Be Bypassed In verify_origin Function .....	17
Lack Of Validation In update_address Function In App Contract.....	18
Inflexibility In Reward Token Management Of Staking Contract .....	19
Centralization Of The Protocol .....	20
Lack Of Control Over Deposited CW721 In Auction Contract .....	21
Lack Of denom Validation In Auction Contract .....	22
Lack Of Token Validation In Crowfund.....	23
Lack Of Kernel Address Validation In Contracts .....	24
Unnecessary Function Inheritance In The Protocol.....	25
Inability To Update The Kernel Address Prevents Incidence Response .....	26
Inadequate Pricing Mechanism In Token Sale .....	27
Limit Of App Components Can Be Bypassed .....	28
Missing Two-Step Ownership Transfer .....	29
Possibility To Start Sale With The Same Assets.....	30
Inefficient Token Purchase Handling In Crowfund .....	31
Potential Fund Loss Due To Overpayment In Marketplace .....	32
Redundant Calculation In Lockdrop Contract.....	33
Querying State Of lockdrop With Time In Past Might Return Incorrect Data.....	34
Unnecessary Usage Of Milliseconds In Auction Contract .....	35
<b>A Proof of Concepts</b>	<b>36</b>

# Executive Summary

**Andromeda Protocol** contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between October 26, 2023 and November 16, 2023. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 3 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 20 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide Andromeda Protocol with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.

The audit process revealed several issues in the existing codebase, especially with the protocol flow. The audit's limited scope restricted a full and effective evaluation of the entire code flow, potentially leaving some vulnerabilities and bugs undetected. It is critical to address these foundational issues promptly and comprehensively. Following remediation, it is strongly advised to schedule a comprehensive re-audit of the entire codebase. This holistic approach in the next audit will ensure that the implemented fixes are thoroughly assessed and that the protocol operates securely and efficiently. A complete re-audit will further contribute to enhancing the protocol's reliability, efficiency, and overall security, providing a more accurate and in-depth understanding of the code's integrity.

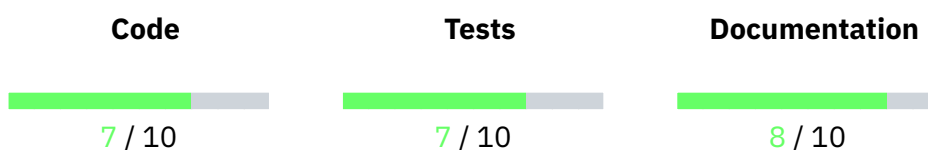


## System Overview

The Andromeda Protocol is a groundbreaking framework and user interface designed to revolutionize the Web 3.0 and blockchain industry with its rapid development capabilities. It offers an "Easier, Better, and Faster" approach, streamlining the process of creating and managing digital assets and applications in the blockchain space utilizing the CosmWasm.



## Repository Coverage and Quality



Resonance's testing team has assessed the Code, Tests, and Documentation coverage and quality of the system and achieved the following results:

- The code follows development best practices and makes use of known patterns, standard libraries, and language guides. It is easily readable but does not use the latest stable version of relevant components. Overall, **code quality is good**.
- Unit and integration tests are included. The tests cover both technical and functional requirements. Code coverage is undetermined. Overall, **tests coverage and quality is good**.
- The documentation includes the specification of the system, technical details for the code, relevant explanations of workflows and interactions. Overall, **documentation coverage and quality is good**.

# Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: [andromedaprotocol/andromeda-core/tree/freeze/audit-2/contracts](https://github.com/andromedaprotocol/andromeda-core/tree/freeze/audit-2/contracts)
- Hash: d320af151238c72e0160529f1dc852944446c93a

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- ecosystem/
- finance/
- os/
- data-storage/

# Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

## Source Code Review - Rust CosmWasm

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

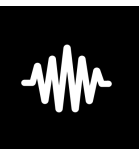
1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Rust CosmWasm audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Frontrunning attacks
- Unsafe third party integrations
- Denial of service
- Access control issues
- Inaccurate business logic implementations
- Incorrect gas usage



- Arithmetic issues
- Unsafe callbacks
- Timestamp dependence
- Mishandled panics, errors and exceptions



## Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

		Likelihood		
		Very Likely	Likely	Unlikely
Impact	Strong	Critical	High	Medium
	Medium	High	Medium	Low
	Weak	Medium	Low	Info





# Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

# Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- ||||| "Quick Win" Requires little work for a high impact on risk reduction.
- |||| "Standard Fix" Requires an average amount of work to fully reduce the risk.
- ||| "Heavy Project" Requires extensive work for a low impact on risk reduction.

Findings ID	Description	Severity	Status
RES-01	Lack Of Module Registration Enforcement Mechanism In Contracts		Resolved
RES-02	Inefficient Validation In Modules Registration Process		Resolved
RES-03	Incorrect Calculation Of Asset To Send In Sale Ratio Calculation		Resolved
RES-04	The Unfair Advantage Of The First Staker		Resolved
RES-05	Lockdrop Participants Won't Be Able To Withdraw Native Assets		Unresolved
RES-06	origin Of The Packet Can Be Bypassed In verify_origin Function		Acknowledged
RES-07	Lack Of Validation In update_address Function In App Contract		Resolved
RES-08	Inflexibility In Reward Token Management Of Staking Contract		Acknowledged
RES-09	Centralization Of The Protocol		Acknowledged
RES-10	Lack Of Control Over Deposited CW721 In Auction Contract		Resolved
RES-11	Lack Of denom Validation In Auction Contract		Resolved
RES-12	Lack Of Token Validation In Crowfund		Acknowledged

<b>RES-13</b>	Lack Of Kernel Address Validation In Contracts		Acknowledged
<b>RES-14</b>	Unnecessary Function Inheritance In The Protocol		Unresolved
<b>RES-15</b>	Inability To Update The Kernel Address Prevents Incidence Response		Resolved
<b>RES-16</b>	Inadequate Pricing Mechanism In Token Sale		Acknowledged
<b>RES-17</b>	Limit Of App Components Can Be Bypassed		Resolved
<b>RES-18</b>	Missing Two-Step Ownership Transfer		Resolved
<b>RES-19</b>	Possibility To Start Sale With The Same Assets		Resolved
<b>RES-20</b>	Inefficient Token Purchase Handling In Crowdfund		Resolved
<b>RES-21</b>	Potential Fund Loss Due To Overpayment In Marketplace		Resolved
<b>RES-22</b>	Redundant Calculation In Lockdrop Contract		Resolved
<b>RES-23</b>	Querying State Of lockdrop With Time In Past Might Return Incorrect Data		Resolved
<b>RES-24</b>	Unnecessary Usage Of Milliseconds In Auction Contract		Resolved



# Lack Of Module Registration Enforcement Mechanism In Contracts

High

RES-ANDR-PRO01

Business Logic

Resolved

## Code Section

- `contracts/fungible-tokens/andromeda-cw20/src/contract.rs#L85`
- `contracts/fungible-tokens/andromeda-cw20-staking/src/contract.rs#L107`
- `contracts/fungible-tokens/andromeda-lockdrop/src/contract.rs#L11`
- `contracts/non-fungible-tokens/andromeda-auction/src/contract.rs#L85`
- `contracts/non-fungible-tokens/andromeda-crowdfund/src/contract.rs#L111`
- `contracts/non-fungible-tokens/andromeda-cw721/src/contract.rs#L286`
- `contracts/non-fungible-tokens/andromeda-marketplace/src/contract.rs#L76`

## Description

The instantiation process in contracts does not mandate the inclusion of modules when initializing the contract. As a result, while it's possible to instantiate a contract without modules, such a contract would essentially be non-functional. This is because each invocation of the `handle_execute` in smart contracts function would attempt to call `contract.module_hook`, leading to an error due to the absence of modules.

## Recommendation

It is recommended to modify the instantiation process in contracts to require the inclusion of modules during the initialization phase. This change would ensure that a contract cannot be instantiated without the necessary modules, preventing the creation of non-functional contracts.

To achieve this, the instantiation logic should include a validation check that confirms the presence of all necessary module before allowing the contract to be deployed.

Furthermore, enhancing the error messaging or handling mechanisms within the `handle_execute` function to provide clear notifications when `contract.module_hook` is called without the required modules could also be beneficial. This approach will improve the robustness of the contract creation process, reducing the likelihood of errors and ensuring that all deployed contracts are functional and equipped with the necessary modules.

## Status

*The issue has been fixed in `a9830aec41ad317a2fd95f392bc626f8c2f0ebee`.*



# Inefficient Validation In Modules Registration Process

High

RES-ANDR-PRO02

Data Validation

Resolved

## Code Section

- `contracts/fungible-tokens/andromeda-cw20/src/contract.rs#L85`
- `contracts/fungible-tokens/andromeda-cw20-staking/src/contract.rs#L107`
- `contracts/fungible-tokens/andromeda-lockdrop/src/contract.rs#L11`
- `contracts/non-fungible-tokens/andromeda-auction/src/contract.rs#L85`
- `contracts/non-fungible-tokens/andromeda-crowdfund/src/contract.rs#L111`
- `contracts/non-fungible-tokens/andromeda-cw721/src/contract.rs#L286`
- `contracts/non-fungible-tokens/andromeda-marketplace/src/contract.rs#L76`

## Description

The current implementation of the module registration process in the protocol's smart contracts exhibits inefficiencies, particularly in the `validate_module_address` function. Modules are crucial for the proper functioning of some smart contracts within the protocol. The `execute_register_module` function permits the owner or operator of the contract to register a module. However, the issue lies in the validation process preceding this registration.

The function `validate_module_address`, which is invoked before `execute_register_module`, is found to be inefficient and potentially flawed. Due to its current design, there is a risk that erroneous or malicious modules could be registered. This inefficiency in validation poses a significant threat to the integrity and functionality of the smart contracts. If a malicious or faulty module is registered, it can render the associated smart contract non-operational or lead to unintended behavior, compromising the reliability and security of the protocol.

The severity of this finding has been escalated to high due to the failure of the deregistration process for erroneous modules.

## Recommendation

It is recommended to enhance and strengthen the `validate_module_address` function to ensure rigorous and effective validation of modules before their registration through `execute_register_module`. This improvement should focus on robustly detecting and rejecting erroneous or malicious modules, thereby safeguarding the integrity and functionality of the smart contracts.

## Status

*The issue has been fixed in 40fe3a6a2489dd36c6d915ab0d9d8710446e4557.*



# Incorrect Calculation Of Asset To Send In Sale Ratio Calculation

High

RES-ANDR-PRO03

Arithmetic Issues

Resolved

## Code Section

- [contracts/fungible-tokens/andromeda-cw20-exchange/src/contract.rs#L273](#)

## Description

A critical vulnerability has been identified in the asset exchange mechanism of the contract, specifically within the function handling sale ratios and asset distribution.

This vulnerability is triggered when a sale ratio and corresponding asset amounts lead to an improper calculation of token disbursement and remainder balance.

For instance, with a sale ratio of 9 and an asset input of 100, the mechanism calculates the number of tokens to be purchased using the formula:  $\text{amount\_sent} / \text{exchange\_rate}$ , which equals  $100/9$ , yielding approximately 11.1111. This result is rounded down to 11. The mechanism then calculates the remainder as  $\text{amount\_sent} - (\text{purchased\_tokens} \times \text{exchange\_rate})$ , which should be  $100 - (11 \times 9) = 1$ . This remainder is meant to be returned to the user, with the original amount\_sent minus the remainder sent to the contract owner.

However, the vulnerability arises because the contract does not account for the extracted remainder in the amount sent to the contract owner, attempting to subtract the total amount\_sent (100) from the remaining balance post-distribution (99).

This results in an overflow error, as the system cannot subtract a larger amount from a smaller one, leading to potential contract failure or incorrect asset distribution. This finding necessitates immediate attention to prevent exploitation and ensure the integrity of the transaction mechanism.

## Recommendation

It is recommended to reimplement the logic of `execute_purchase` to correctly handle the distribution of the assets.

## Status

*This issue has been fixed in 110d4a57afd0a7b4f2c26aedaf718c78fd54e6bc.*



# The Unfair Advantage Of The First Staker

High

RES-ANDR-PRO04

Business Logic

Resolved

## Code Section

- `contracts/fungible-tokens/andromeda-cw20-staking/src/contract.rs#L596`

## Description

The `contracts/fungible-tokens/andromeda-cw20-staking/src/contract.rs` is designed to permit users to stake assets selected by the contract owner, with rewards calculated based on the shares held by each staker. However, when incentives are deposited into the contract prior to any staking transactions, then the first user to call the `update_staker_reward_info` function is able to claim all these deposited incentives for themselves when `AllocationConfig` is not set.

## Recommendation

It is recommended to revise the function's logic to ensure accurate calculation of rewards.

## Status

*The issue has been fixed in `7c89d5eb3a707d1157e04a135958d8b772c61fca`.*





# Lockdrop Participants Won't Be Able To Withdraw Native Assets

High

RES-ANDR-PR005

Business Logic

Unresolved

## Code Section

- [contracts/fungible-tokens/andromeda-lockdrop/src/contract.rs#L299](#)

## Description

The `andromeda-lockdrop/src/contract.rs` enables users to lock native coins in the contract, receiving incentive tokens in return. During a predefined withdrawal window, users have the option to partially withdraw their deposited assets. However, once the lockdrop period concludes, users lose the ability to withdraw their locked assets, only the contract owner retains this capability.

## Recommendation

It is recommended to modify the `andromeda-lockdrop/src/contract.rs` to allow users to withdraw their locked assets even after the lockdrop period has concluded. This change would enhance user trust and autonomy, ensuring that users are not solely reliant on the contract owner for the retrieval of their assets. Implementing a secure and transparent withdrawal mechanism post-lockdrop can significantly improve the contract's functionality and user experience.



# origin Of The Packet Can Be Bypassed In verify\_origin Function

High

RES-ANDR-PRO06

Data Validation

Acknowledged

## Code Section

- All contracts

## Description

The `verify` function inside the `execute_amp_receive` method includes a sequence of checks to validate `info.sender`. Initially, it verifies if `info.sender` matches either `self.ctx.origin` or `kernel_address`. However, the validity of this check is compromised as `self.ctx.origin` can be manipulated by `info.sender`, rendering the check potentially ineffective. In cases where this initial check fails, the function then verifies if `info.sender`'s `codeid` is listed in the ADODB. This too is flawed, as any user can instantiate a contract with the specified `codeid`, allowing this check to be bypassed.

Additionally, these initial verifications are rendered redundant by subsequent validations in the `execute` functions. These later checks more rigorously assess whether the sender has the necessary permissions to perform the action, thereby superseding the earlier, less secure validations.

The significance of this finding has been diminished, as crucial validation is adequately addressed in subsequent functions that are executed.

## Recommendation

It is recommended to reevaluate and potentially remove the initial sender verification checks, given their susceptibility to manipulation and redundancy.

Strengthen and rely on the subsequent, more comprehensive permission checks in the `execute` functions to ensure secure and appropriate validation of `info.sender`.

By simplifying the verification process and concentrating on the more secure checks, the protocol can improve its security posture and reduce unnecessary code complexity.

## Status

*The issue has been acknowledged by Andromeda team. The development team stated "Origin field cannot be maliciously overridden if authorisation of sending the message is restricted to validated contracts. The validation that no malicious activity is undertaken is done before the contract is authorised in the ADODB."*



# Lack Of Validation In `update_address` Function In App Contract

Medium RES-ANDR-PRO07

Data Validation

Resolved

## Code Section

- [contracts/app/andromeda-app-contract/src/execute.rs#L15](#)

## Description

The `update_address` function within the contract lacks validation of the `ado_type` and the existence of the provided address. Without these validations, an incorrect or non-existent address entered mistakenly could lead to unintended and potentially problematic behavior within the protocol.

## Recommendation

It is recommended to implement a comprehensive validation mechanism for the `ado_type` and the address within the `update_address` function. This should include checks to confirm the validity of the `ado_type` and the existence of the provided address.

## Status

*The issue has been fixed in `dbb8e9b11211cb1f63c3e131b4a250c53204f8a1`.*



# Inflexibility In Reward Token Management Of Staking Contract

Medium RES-ANDR-PRO08

Business Logic

Acknowledged

## Code Section

- [contracts/fungible-tokens/andromeda-cw20-staking/src/contract.rs#L131](#)

## Description

The `execute_add_reward_token` in `contracts/fungible-tokens/andromeda-cw20-staking/src/contract.rs` currently does not offer a mechanism to remove or replace a reward token once it has been set. This lack of functionality presents a risk in scenarios where an incorrect token is provided or if there is a need to update the reward token due to changes in the project's tokenomics or other strategic decisions.

## Recommendation

Implement functionality that allows contract administrators to remove or replace the reward token. This change would increase the flexibility and robustness of the contract, thereby enhancing operational security and adaptability to future needs.

## Status

*The issue has been acknowledged by Andromeda team.*



# Centralization Of The Protocol

Medium RES-ANDR-PR009

Business Logic

Acknowledged

## Code Section

- Not Specified

## Description

The protocol exhibits a notable degree of centralization, primarily due to its heavy reliance on the kernel, which is identified as the most critical contract in the system. This centralization raises several concerns, particularly in light of previously identified vulnerabilities in the kernel, such as the lack of address validation and the inability to update the kernel address post-deployment.

The kernel's pivotal role in the protocol means that virtually all operations and functionalities are dependent on it. This over-reliance creates a single point of failure, where any compromise or malfunction in the kernel could have far-reaching and detrimental effects on the entire system. The risks are exacerbated by the identified vulnerabilities, which can potentially be exploited, leading to system-wide disruptions or security breaches.

## Recommendation

It is recommended to decentralize the reliance on the kernel by distributing responsibilities and control mechanisms across multiple contracts within the protocol. This approach would mitigate the risks associated with having a single point of failure and enhance the overall resilience of the system.

## Status

*The issue has been acknowledged by the Andromeda team. The development team stated "Agree with this point, would require larger architectural changes which would need to come post launch."*



# Lack Of Control Over Deposited CW721 In Auction Contract

Medium RES-ANDR-PRO10

Data Validation

Resolved

## Code Section

- [contracts/non-fungible-tokens/andromeda-auction/src/contract.rs#L154](#)

## Description

The method `execute_start_auction` in the specified contract allows users to deposit CW721 tokens to initiate auctions. However, it currently lacks controls to validate or restrict the types of CW721 tokens that can be used. This oversight could potentially be exploited by malicious actors, who may initiate numerous auctions using illegitimate or malicious tokens.

Allowing illegitimate tokens could negatively impact the platform's credibility and user trust moreover, without validation, there's a risk of storage bloating, where the contract's storage could be overwhelmed with data from numerous auctions of malicious or spam tokens. This can lead to increased costs and performance degradation, potentially rendering the contract inefficient or even unusable over time.

## Recommendation

To mitigate these risks, it's advised to implement a validation mechanism for CW721 tokens within the `execute_start_auction` method. This could involve:

- Creating a list of approved tokens and checking deposited tokens against this list.
- Implementing a verification process for new tokens before they are allowed in auctions.
- Regularly updating and auditing the list to ensure ongoing security and legitimacy.

## Status

*The issue has been fixed in 408657413545aa518e2cff85aeca92ca0e80bad2.*



# Lack Of denom Validation In Auction Contract

Medium RES-ANDR-PRO11

Data Validation

Resolved

## Code Section

- [contracts/non-fungible-tokens/andromeda-auction/src/contract.rs#L141](#)

## Description

The `execute_start_auction` and `execute_update_auction` methods in the `contracts/non-fungible-tokens/andromeda-auction/src/contract.rs` allow users to initiate or update auctions by providing a CW721 asset and basic auction details, including `coin_denom`. Presently, the `coin_denom` parameter is accepted as a string in both methods, but there is no mechanism in place to validate the legitimacy or correctness of the provided denomination.

## Recommendation

It is recommended to implement a robust validation system for the `coin_denom` parameter in both `execute_start_auction` and `execute_update_auction` methods.

## Status

*The issue has been fixed in 076ffb93a731479ef2a8183b89e1785108808f87.*





# Lack Of Token Validation In Crowfund

Medium RES-ANDR-PRO12

Data Validation

Acknowledged

## Code Section

- `contracts/non-fungible-tokens/andromeda-crowdfund/src/contract.rs#L50`

## Description

The `instantiate` function within the `contracts/non-fungible-tokens/andromeda-crowdfund/src/contract.rs` file exhibits a vulnerability due to its failure to validate the `token_address` parameter. This oversight allows the `CONFIG.save()` method to be executed even if an invalid `token_address` is provided.

The function does not include a mechanism to verify the validity of the `token_address` being input. This poses a significant risk as it could lead to the configuration being saved with an incorrect or malicious address. Moreover, the contract lacks a function to update the configuration. This limitation becomes particularly problematic in the context of the validation issue. If an incorrect `token_address` is saved, there is no direct method to correct this error within the existing deployment.

## Recommendation

It is recommended to implement a validation mechanism for the `token_address` within the `instantiate` function of the `contracts/non-fungible-tokens/andromeda-crowdfund/src/contract.rs` file. This will ensure that `CONFIG.save()` is not called with an invalid `token_address`, thereby enhancing the security and integrity of the contract.

## Status

*The issue has been acknowledged by the Andromeda team.*



# Lack Of Kernel Address Validation In Contracts

Medium RES-ANDR-PRO13

Data Validation

Acknowledged

## Code Section

- All contracts

## Description

This finding highlights a significant oversight in the smart contract architecture related to the lack of kernel address validation. The kernel, being the most crucial contract in the protocol, requires rigorous validation to ensure its integrity and correct functioning. However, it has been observed that there is no mechanism in place to validate whether the correct contract address has been provided in any of the smart contracts. This absence of validation poses a substantial risk, as the entry of an incorrect address can lead to critical operational failures.

Furthermore, the current design lacks a mechanism to update or change the kernel address once it is set. This inflexibility means that if a wrong address is mistakenly provided, there is no recourse other than to redeploy the entire protocol. This not only presents a significant inconvenience but also poses a risk to the stability and reliability of the system. The absence of an address update mechanism coupled with the lack of initial validation creates a vulnerability that could potentially jeopardize the entire protocol's functionality.

## Recommendation

It is recommended to incorporate stringent validation processes for the kernel address in all smart contracts within the protocol. This validation should ensure that only the correct kernel address is used, thereby preventing operational failures due to incorrect address inputs. Additionally, it is advisable to develop and integrate a mechanism that allows for the updating or modification of the kernel address post-deployment. This feature would provide a contingency plan to rectify any mistakes in address input without necessitating a complete redeployment of the protocol. Implementing these measures will significantly enhance the security, flexibility, and overall robustness of the system.

## Status

*The issue has been acknowledged by the Andromeda team.*



# Unnecessary Function Inheritance In The Protocol

Medium RES-ANDR-PR014

Business Logic

Unresolved

## Code Section

- All contracts

## Description

In the protocol's smart contract structure, there is an inheritance pattern where contracts inherit `_ => ADOContract::default().execute(ctx, msg)`. This inheritance brings with it additional `execute` functions from `ADOContract::default()`. While these additional functions are designed to enhance the capabilities of the smart contracts, some of these inherited functions are not useful or relevant to the given smart contract.

The inclusion of unnecessary `execute` functions presents a two-fold problem. First, it introduces unnecessary attack vectors. Functions that are not relevant to the contract's intended operation but are still accessible can be exploited by malicious actors, potentially leading to security breaches.

Second, the presence of these superfluous functions leads to expanded usage of the storage. Storage space is a precious resource in smart contract environments, and its efficient use is paramount. By inheriting and retaining functions that are not needed, the smart contract inadvertently consumes more storage than necessary. This not only affects the efficiency and performance of the contract itself but could also have broader implications on the overall resource management within the environment.

## Recommendation

It is recommended to carefully review and modify the smart contract's inheritance structure to ensure that only necessary and relevant `execute` functions from `ADOContract::default()` are included. This process should involve assessing each inherited function for its utility and relevance to the specific requirements of the smart contract. Functions that are not essential should be excluded to eliminate unnecessary attack vectors and reduce the use of storage. Additionally, implementing a more selective inheritance strategy or providing an option to override and disable irrelevant functions could further streamline the contract's functionality and enhance its security and efficiency. This targeted approach will not only optimize resource usage but also strengthen the contract's defense against potential exploits.



# Inability To Update The Kernel Address Prevents Incidence Response

Medium RES-ANDR-PR015

Business Logic

Resolved

## Code Section

- Not Specified

## Description

The current architecture of the smart contract system exhibits the inability to update the kernel address post-deployment. This limitation poses a significant risk, particularly in scenarios that require a rapid response to incidents. In an ideal setup, the flexibility to modify the kernel address is crucial for effective incident response strategies, such as addressing security breaches or updating the system to rectify operational inefficiencies.

Without the capability to update the kernel address, the system is locked into its initial configuration, making it rigid and unresponsive to emerging challenges or threats. This rigidity can lead to prolonged downtime or even catastrophic failure in the event of a critical issue that requires an immediate change in the kernel address.

## Recommendation

It is recommended to introduce a feature in the smart contract system that allows for the dynamic updating of the kernel address. This update mechanism should be designed with strict security measures to prevent unauthorized access while enabling authorized users to respond swiftly to incidents or operational changes. Implementing this feature will significantly enhance the system's adaptability and resilience, allowing for a more efficient response to security breaches, bugs, or other critical issues that may arise. Moreover, this flexibility in managing the kernel address is crucial for maintaining the protocol's integrity and reliability in a rapidly evolving digital environment.

## Status

*The issue has been fixed in 29434ecb19fc2ed8afd33d36ae07de5f86714d21.*



# Inadequate Pricing Mechanism In Token Sale

Medium RES-ANDR-PR016

Business Logic

Acknowledged

## Code Section

- `contracts/fungible-tokens/andromeda-cw20-exchange/src/contract.rs#L312`

## Description

The `purchase` function in `contracts/fungible-tokens/andromeda-cw20-exchange/src/contract.rs` allow users to buy tokens from sale. However there is no mechanism implemented for scenarios where the value of the sold token may exceed that of the token being received in exchange.

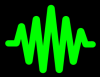
The implication of this oversight is that the smart contract does not prevent transactions in which users could unintentionally sell their tokens at a loss, because it fails to dynamically adjust the sale price in response to fluctuating market values. This could lead to financial losses for sellers and potentially undermine the platform's credibility.

## Recommendation

It is recommended to implement a dynamic pricing mechanism. This mechanism should adjust the sale price of tokens in real-time or near-real-time to reflect current market values.

## Status

*The issue has been acknowledged by the Andromeda team.*



# Limit Of App Components Can Be Bypassed

Low

RES-ANDR-PRO17

Data Validation

Resolved

## Code Section

- `contracts/app/andromeda-app-contract/src/execute.rs#L18`

## Description

The instantiation function within `contracts/app/andromeda-app-contract/src/execute.rs` has restriction to register a maximum of 50 app components. However, this restriction can be bypassed during the execution of the `handle_add_app_component` function.

## Recommendation

It is recommended to revise the `handle_add_app_component` function in `contracts/app/andromeda-app-contract/src/execute.rs` to enforce the same limit of 50 app components as set in the instantiation function. This could involve adding a check within `handle_add_app_component` to count the current number of components and prevent the addition of new components once the limit of 50 is reached. Implementing this consistency ensures that the application adheres to the predefined limits, maintaining efficiency and preventing potential performance issues due to an excessive number of components.

## Status

*The issue has not been fixed in 3bfb7be6bb8eb0f5fdef276d3f04b976cf35c17a.*



# Missing Two-Step Ownership Transfer

Low

RES-ANDR-PR018

Access Control

Resolved

## Code Section

- All contracts

## Description

The protocol currently lacks a two-step process for ownership transfer.

In many secure contract systems, ownership transfer involves a two-phase approach where the new owner must explicitly accept ownership after the current owner initiates the transfer. This two-step process is a critical safeguard against unauthorized or accidental changes in ownership.

In the existing setup of the protocol, ownership transfer appears to be a single-step action, which significantly increases the risk of unintended ownership changes. This can happen due to human error, such as inputting an incorrect address, or through malicious activities, where an attacker might exploit this vulnerability to seize control of the contract.

## Recommendation

It is recommended to implement a two-step ownership transfer process in the protocol. This process should require the new owner to explicitly accept ownership after the current owner initiates the transfer. Such a mechanism ensures an additional layer of security, verifying that both parties consent to the transfer and reducing the risk of unauthorized or accidental changes in ownership.

## Status

*The issue has been fixed in 3bfb7be6bb8eb0f5fdef276d3f04b976cf35c17a.*





# Possibility To Start Sale With The Same Assets

Low

RES-ANDR-PRO19

Data Validation

Resolved

## Code Section

- `contracts/fungible-tokens/andromeda-cw20-exchange/src/contract.rs#L134`

## Description

In the Andromeda CW20 Exchange, specifically within the file `contracts/fungible-tokens/andromeda-cw20-exchange/src/contract.rs`, an issue has been identified with the `execute_start_sale` function. This function currently allows the contract owner to initiate a sale using the same pair of tokens.

Under normal circumstances, a token sale should involve two different tokens — one being sold and the other being used for purchase. However, the ability for the contract owner to start a sale with the same token for both roles creates a scenario that is not only logically inconsistent but could also lead to confusion and potential misuse within the platform.

This functionality could enable scenarios where tokens are being exchanged for themselves, which defies the standard trading or exchange logic and could result in unexpected behavior in the contract. It might also open up possibilities for manipulating the market or token values in a way that could be detrimental to the integrity of the exchange and its users.

## Recommendation

It is recommended to modify the `execute_start_sale` function in `contracts/fungible-tokens/andromeda-cw20-exchange/src/contract.rs` to include validation checks that prevent the initiation of sales with identical token pairs. This can be achieved by implementing a logic check within the function that verifies the uniqueness of the tokens involved in the sale.

## Status

*The issue has been fixed in `0a506ef7b81e600de1f3f09320ebda2188e40878`.*



# Inefficient Token Purchase Handling In Crowdfund

Low

RES-ANDR-PRO20

Gas Optimization

Resolved

## Code Section

- Not specified

## Description

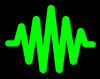
The `execute_purchase` function in the `contracts/non-fungible-tokens/andromeda-crowdfund/src/contract.rs` smart contract allows users to purchase tokens. Currently, if a user attempts to buy more tokens than the set purchase limit, the contract processes the transaction by allocating only the maximum allowable amount of tokens to the user, and the excess funds are returned. This approach, while functional, is inefficient in terms of gas usage and transaction processing time.

## Recommendation

It would be more efficient to incorporate a preliminary check within the `execute_purchase` function to immediately identify when a user's requested token amount exceeds the purchase limit. In such cases, instead of partially processing the transaction and refunding the excess amount, the transaction should be aborted, and an error should be returned to the user. This change would optimize the contract by preventing unnecessary computation and reducing the potential for wasted gas fees, enhancing the overall user experience and contract performance.

## Status

*The issue has been fixed in `f7bf815c3fc13e3aa12ea474fa3ec9e959109392`.*



# Potential Fund Loss Due To Overpayment In Marketplace

Low

RES-ANDR-PRO21

Data Validation

Resolved

## Code Section

- [contracts/non-fungible-tokens/andromeda-marketplace/src/contract.rs#L276](#)

## Description

The `execute_buy` function in the Andromeda Marketplace smart contract facilitates the purchase of tokens by users. The current implementation of this function checks if `payment.amount` is greater than or equal to `token_sale_state.price`. However, this implementation does not handle scenarios where a user accidentally sends an amount greater than the required price. In such cases, the excess funds sent by the user are not refunded, leading to a potential loss of funds.

## Recommendation

It is recommended to modify the `execute_buy` function to include a mechanism that not only checks for the minimum required amount but also handles overpayment scenarios.

## Status

*The issue has been fixed in [3bfb7be6bb8eb0f5fdef276d3f04b976cf35c17a](#)*



# Redundant Calculation In Lockdrop Contract

Info

RES-ANDR-PRO22

Arithmetic Issues

Resolved

## Code Section

- [contracts/fungible-tokens/andromeda-lockdrop/src/contract.rs#L600](#)

## Description

The `allowed_withdrawal_percent` in `contracts/fungible-tokens/andromeda-lockdrop/src/contract.rs` determine the percentage of withdrawals allowed within a specified time frame. It contains a conditional statement that checks if the current timestamp is less than a predefined `withdrawal_cutoff_init_point`. If this condition is true, the function returns a percentage value.

The issue here lies in the return statement: `return Decimal::from_ratio(100u32, 100u32);`. This line of code is intended to return a decimal representation of 100%, indicating full withdrawal allowance. However, the calculation is redundant since it involves dividing 100 by 100, which is an unnecessary operation as it always yields 1 or 100%.

## Recommendation

It's recommended to simplify this calculation to enhance code clarity and maintainability. The current implementation of the withdrawal percentage calculation in the deposit window function involves a redundant operation.

## Status

*The issue has been fixed in `f7bf815c3fc13e3aa12ea474fa3ec9e959109392`.*



# Querying State Of lockdrop With Time In Past Might Return Incorrect Data

Info RES-ANDR-PRO23 Timestamp Dependence

Resolved

## Code Section

- [contracts/fungible-tokens/andromeda-lockdrop/src/contract.rs#L56](#)

## Description

The `query_max_withdrawable_percent` function is intended to enable users to determine the maximum percentage of funds they can withdraw at a given time. A critical oversight in this function is its failure to validate whether the provided timestamp is in the past.

## Recommendation

Implement a validation step within the function to ensure that the provided timestamp is either current or future-dated. If the timestamp is found to be in the past, the function should handle this appropriately, either by returning an error or a default value that clearly indicates the inapplicability of past timestamps for withdrawal calculations.

## Status

*The issue has been fixed in 07be58a5432e2fb34bec1b88543bbdef59ce6ec9.*



# Unnecessary Usage Of Milliseconds In Auction Contract

Info

RES-ANDR-PRO24

Timestamp Dependence

Resolved

## Code Section

- [contracts/non-fungible-tokens/andromeda-auction/src/contract.rs#L173](#)

## Description

The `contracts/non-fungible-tokens/andromeda-auction/src/contract.rs` contract includes a conversion of time units from nanoseconds to milliseconds. This conversion is performed in the functions `expiration_from_nanoseconds()`. It appears that this conversion from nanoseconds to milliseconds is not required for the auction's functionality. The precision offered by milliseconds is not utilized and all time-sensitive interactions, including bid timings and auction duration, can be adequately handled in seconds.

## Recommendation

It is recommended to eliminate the unnecessary conversion of time units from nanoseconds to milliseconds in the smart contract handling the auction. This simplification will not only reduce the complexity of the code but also mitigate potential risks associated with time conversion errors.

## Status

*The issue has been fixed in `a186133124000720c8ea9d9ce2e2eceb77429077`.*

# Proof of Concepts

## RES-01 Lack Of Module Registration Enforcement Mechanism In Contracts

```
#[test]
fn test_exploit(){
    let mut app = App::default();

    let code = ContractWrapper::new(cw721_execute, cw721_instantiate, cw721_query);
    let cw_721_code_id = app.store_code(Box::new(code));

    let cw721_init = cw721_InstantiateMsg{ name: "NFT".to_owned(), symbol:
↳ "NFT".to_owned(), minter: "owner".to_owned() };

    let cw721_addr = app.instantiate_contract(cw_721_code_id,
↳ Addr::unchecked("owner"), &cw721_init, &[], "nft", None).unwrap();
    //instantiate kernel

    let init_msg = InstantiateMsg {
        owner: None,
        modules: None, //no modules needed
        kernel_address: MOCK_KERNEL_CONTRACT.to_string(),
    };

    let code = ContractWrapper::new(execute, instantiate, query);
    let auction_code_id = app.store_code(Box::new(code));

    let auction_addr = app.instantiate_contract(auction_code_id,
↳ Addr::unchecked("owner"), &init_msg, &[], "staking", None).unwrap();
}
```

## RES-02 Inefficient Validation In Modules Registration Process

```
use andromeda_app::app::{AppComponent, ComponentType};
use andromeda_app_contract::mock::{
    mock_andromeda_app, mock_app_instantiate_msg, mock_claim_ownership_msg,
↳ mock_get_address_msg,
    mock_get_components_msg,
};
use andromeda_crowdfund::mock::{
    mock_andromeda_crowdfund, mock_crowdfund_instantiate_msg,
↳ mock_crowdfund_quick_mint_msg,
    mock_end_crowdfund_msg, mock_purchase_msg, mock_start_crowdfund_msg,
};
use andromeda_cw721::mock::{
    mock_andromeda_cw721, mock_cw721_instantiate_msg, mock_cw721_owner_of,
};
use andromeda_finance::splitter::AddressPercent;
use andromeda_non_fungible_tokens::crowdfund::ExecuteMsg;
use andromeda_std::amp::{AndrAddr, Recipient};
```



```

use andromeda_modules::rates::{Rate, RateInfo};
use andromeda_rates::mock::{mock_andromeda_rates, mock_rates_instantiate_msg};
use andromeda_splitter::mock::{
    mock_andromeda_splitter, mock_splitter_instantiate_msg, mock_splitter_send_msg,
};
use andromeda_std::ado_base::modules::Module;
use std::str::FromStr;

use andromeda_testing::mock::MockAndromeda;
use andromeda_vault::mock::{
    mock_andromeda_vault, mock_vault_deposit_msg, mock_vault_instantiate_msg,
};
use cosmwasm_std::{coin, to_binary, Addr, BlockInfo, Decimal, Uint128};
use cw721::{Expiration, OwnerOfResponse};
use cw_multi_test::{App, Executor};

fn mock_app() -> App {
    App::new(|router, _api, storage| {
        router
            .bank
            .init_balance(
                storage,
                &Addr::unchecked("owner"),
                [coin(999999, "uandr")].to_vec(),
            )
            .unwrap();
        router
            .bank
            .init_balance(
                storage,
                &Addr::unchecked("buyer_one"),
                [coin(100, "uandr")].to_vec(),
            )
            .unwrap();
        router
            .bank
            .init_balance(
                storage,
                &Addr::unchecked("buyer_two"),
                [coin(100, "uandr")].to_vec(),
            )
            .unwrap();
        router
            .bank
            .init_balance(
                storage,
                &Addr::unchecked("buyer_three"),
                [coin(100, "uandr")].to_vec(),
            )
            .unwrap();
    })
}

```

```

}

fn mock_andromeda(app: &mut App, admin_address: Addr) -> MockAndromeda {
    MockAndromeda::new(app, &admin_address)
}

#[test]
fn test_crowdfund_app() {
    let owner = Addr::unchecked("owner");
    let vault_one_recipient_addr = Addr::unchecked("vault_one_recipient");
    let vault_two_recipient_addr = Addr::unchecked("vault_two_recipient");
    let buyer_one = Addr::unchecked("buyer_one");
    let buyer_two = Addr::unchecked("buyer_two");
    let buyer_three = Addr::unchecked("buyer_three");

    let mut router = mock_app();
    let andr = mock_andromeda(&mut router, owner.clone());

    // Store contract codes
    let cw721_code_id = router.store_code(mock_andromeda_cw721());
    let crowdfund_code_id = router.store_code(mock_andromeda_crowdfund());
    let vault_code_id = router.store_code(mock_andromeda_vault());
    let splitter_code_id = router.store_code(mock_andromeda_splitter());
    let app_code_id = router.store_code(mock_andromeda_app());
    let rates_code_id = router.store_code(mock_andromeda_rates());

    andr.store_code_id(&mut router, "cw721", cw721_code_id);
    andr.store_code_id(&mut router, "crowdfund", crowdfund_code_id);
    andr.store_code_id(&mut router, "vault", vault_code_id);
    andr.store_code_id(&mut router, "splitter", splitter_code_id);
    andr.store_code_id(&mut router, "app", app_code_id);
    andr.store_code_id(&mut router, "rates", rates_code_id);

    // Generate App Components
    // App component names must be less than 3 characters or longer than 54
    ↪ characters to force them to be 'invalid' as the MockApi struct used within the
    ↪ CosmWasm App struct only contains those two validation checks
    let rates_recipient = "rates_recipient";
    // Generate rates contract
    let rates: Vec<RateInfo> = [RateInfo {
        rate: Rate::Flat(coin(1, "uandr")),
        is_additive: false,
        recipients: [Recipient::from_string(rates_recipient.to_string())].to_vec(),
        description: Some("Some test rate".to_string()),
    }]
    .to_vec();
    let rates_init_msg = mock_rates_instantiate_msg(rates,
    ↪ andr.kernel_address.to_string(), None);
    let rates_addr = router
        .instantiate_contract(
            rates_code_id,
            owner.clone(),

```

```

        &rates_init_msg,
        &[],
        "rates",
        None,
    )
    .unwrap();

let modules: Vec<Module> = vec![Module::new("rates", rates_addr.to_string(),
↪ false)];

let crowdfund_init_msg = mock_crowdfund_instantiate_msg(
    AndrAddr::from_string("./2".to_string()),
    false,
    Some(modules),
    andr.kernel_address.to_string(),
    None,
);
let crowdfund_app_component = AppComponent {
    name: "1".to_string(),
    ado_type: "crowdfund".to_string(),
    component_type: ComponentType::New(to_binary(&crowdfund_init_msg).unwrap()),
};

let cw721_init_msg = mock_cw721_instantiate_msg(
    "Test Tokens".to_string(),
    "TT".to_string(),
    "./1", // Crowdfund must be minter
    None,
    andr.kernel_address.to_string(),
    None,
);
let cw721_component = AppComponent {
    name: "2".to_string(),
    ado_type: "cw721".to_string(),
    component_type: ComponentType::new(cw721_init_msg),
};

let vault_one_init_msg =
↪ mock_vault_instantiate_msg(andr.kernel_address.to_string(), None);
let vault_one_app_component = AppComponent {
    name: "3".to_string(),
    ado_type: "vault".to_string(),
    component_type: ComponentType::new(&vault_one_init_msg),
};

let vault_two_init_msg =
↪ mock_vault_instantiate_msg(andr.kernel_address.to_string(), None);
let vault_two_app_component = AppComponent {
    name: "4".to_string(),
    ado_type: "vault".to_string(),
    component_type: ComponentType::new(&vault_two_init_msg),
};

```

```

// Create splitter recipient structures
let vault_one_recipient =
    Recipient::from_string(format!("{}",
    ↪ vault_one_app_component.name)).with_msg(
        mock_vault_deposit_msg(
            Some(AndrAddr::from_string(vault_one_recipient_addr.to_string())),
            None,
        ),
    );
let vault_two_recipient =
    Recipient::from_string(format!("{}",
    ↪ vault_two_app_component.name)).with_msg(
        mock_vault_deposit_msg(
            Some(AndrAddr::from_string(vault_two_recipient_addr.to_string())),
            None,
        ),
    );

let splitter_recipients = vec![
    AddressPercent {
        recipient: vault_one_recipient,
        percent: Decimal::from_str("0.5").unwrap(),
    },
    AddressPercent {
        recipient: vault_two_recipient,
        percent: Decimal::from_str("0.5").unwrap(),
    },
];

let splitter_init_msg =
    ↪ mock_splitter_instantiate_msg(splitter_recipients,
    andr.kernel_address.clone(), None, None);
let splitter_app_component = AppComponent {
    name: "5".to_string(),
    component_type: ComponentType::new(&splitter_init_msg),
    ado_type: "splitter".to_string(),
};

let app_components = vec![
    cw721_component.clone(),
    crowdfund_app_component.clone(),
    vault_one_app_component.clone(),
    vault_two_app_component.clone(),
    splitter_app_component.clone(),
];

let app_init_msg = mock_app_instantiate_msg(
    "app".to_string(),
    app_components.clone(),
    andr.kernel_address.clone(),
    None,
);

```

```

let app_addr = router
    .instantiate_contract(
        app_code_id,
        owner.clone(),
        &app_init_msg,
        &[],
        "Crowdfund App",
        Some(owner.to_string()),
    )
    .unwrap();

let components: Vec<AppComponent> = router
    .wrap()
    .query_wasm_smart(app_addr.clone(), &mock_get_components_msg())
    .unwrap();

assert_eq!(components, app_components);

let _vault_one_addr: String = router
    .wrap()
    .query_wasm_smart(
        app_addr.clone(),
        &mock_get_address_msg(vault_one_app_component.name),
    )
    .unwrap();

let _vault_two_addr: String = router
    .wrap()
    .query_wasm_smart(
        app_addr.clone(),
        &mock_get_address_msg(vault_two_app_component.name),
    )
    .unwrap();

router
    .execute_contract(
        owner.clone(),
        app_addr.clone(),
        &mock_claim_ownership_msg(None),
        &[],
    )
    .unwrap();

let crowdfund_addr: String = router
    .wrap()
    .query_wasm_smart(
        app_addr.clone(),
        &mock_get_address_msg(crowdfund_app_component.name),
    )
    .unwrap();

```

```

//Human error
let module = Module{ name: Some("random".to_owned()), address:
↪ AndrAddr::from_string("badaddress"), is_mutable: false };
let msg = ExecuteMsg::RegisterModule{ module: module };
let result = router.execute_contract(owner.clone(),
↪ Addr::unchecked(crowdfund_addr.clone()), &msg, &[]).unwrap();

let msg = ExecuteMsg::DeregisterModule { module_idx: 2u64.into() };
let result = router.execute_contract(owner.clone(),
↪ Addr::unchecked(crowdfund_addr.clone()), &msg, &[]).unwrap();
}

```

## RES-03 Incorrect Calculation Of Asset To Send In Sale Ratio Calculation

```

#[test]
fn test_native_exploit(){
    let mut app = App::default();
    let code = ContractWrapper::new(execute_cw20, instantiate_cw20, query_cw20);

    let cw_20_code_id = app.store_code(Box::new(code));

    let cw20_init_msg = cw20_InstantiateMsg{ name: "Token".to_owned(), symbol:
↪ "TOK".to_owned(), decimals: 18u8, initial_balances:
    vec![Cw20Coin {
        amount: 100u128.into(),
        address: "owner".to_string(),
    },
    Cw20Coin {
        amount: 100u128.into(),
        address: "user2".to_string(),
    }
    ], mint: None, marketing: None };

    let cw20_address= app.instantiate_contract(cw_20_code_id,
↪ Addr::unchecked("owner"), &cw20_init_msg, &[], "Token", None).unwrap();

    let code = ContractWrapper::new(execute, instantiate, query);
    let code = ContractWrapper::with_reply(code, crate::contract::reply);
    let exchange_code_id = app.store_code(Box::new(code));

    let init_msg = InstantiateMsg {
        kernel_address: MOCK_KERNEL_CONTRACT.to_string(),
        owner: None,
        modules: None,
        token_address: AndrAddr::from_string(cw20_address.clone().to_string()),
    };

    let exchange_addr = app.instantiate_contract(exchange_code_id,
↪ Addr::unchecked("owner"), &init_msg, &[], "exchange", None).unwrap();

    //CREATE ASSET exchange

```

```

let hook = Cw20HookMsg::StartSale {
    asset: AssetInfo::Native("test".to_string()),
    exchange_rate: Uint128::from(9u128),
    recipient: None,
};

let msg = Cw20ExecuteMsg::Send {
    contract: exchange_addr.to_string(),
    amount: 100u128.into(),
    msg: to_binary(&hook).unwrap()
};

let result = app.execute_contract(Addr::unchecked("owner"),
↪ cw20_address.clone(), &msg, &[]).unwrap();

mint_native(&mut app, "user".to_owned(), "test".to_owned(), 100u128);

//purchase with token

let msg = ExecuteMsg::Purchase { recipient: None};
let result = app.execute_contract(Addr::unchecked("user"),
↪ exchange_addr.clone(), &msg, &[coin(100, "test")]).unwrap();

```

## RES-04 The Unfair Advantage Of The First Staker

```

#[test]
fn test_exploit(){
    let mut app = App::default();
    let code = ContractWrapper::new(execute_cw20, instantiate_cw20, query_cw20);
    let cw_20_code_id = app.store_code(Box::new(code));

    let cw20_init_msg = cw20_InstantiateMsg{ name: "Token".to_owned(), symbol:
↪ "TOK".to_owned(), decimals: 18u8, initial_balances:
    vec![Cw20Coin {
        amount: 20000u128.into(),
        address: "owner".to_string(),
    }
    ], mint: None, marketing: None };

    let cw20_incentives_address= app.instantiate_contract(cw_20_code_id,
↪ Addr::unchecked("owner"), &cw20_init_msg, &[], "Token", None).unwrap();

    let cw20_staking_init_msg = cw20_InstantiateMsg{ name: "Token".to_owned(),
↪ symbol: "TOK".to_owned(), decimals: 18u8, initial_balances:
    vec![Cw20Coin {
        amount: 1000u128.into(),
        address: "user".to_string(),
    }
    ],
    Cw20Coin {
        amount: 1000u128.into(),

```

```

        address: "user2".to_string(),
    },
    Cw20Coin {
        amount: 1000u128.into(),
        address: "user3".to_string(),
    }
], mint: None, marketing: None };

let cw20_staking_token_address = app.instantiate_contract(cw20_code_id,
↪ Addr::unchecked("owner"), &cw20_staking_init_msg, &[], "Token", None).unwrap();

let code = ContractWrapper::new(execute, instantiate, query);
let staking_code_id = app.store_code(Box::new(code));
let current_timestamp = app.block_info().time.seconds();

let additional_rewards = Some(vec![
    RewardTokenUnchecked {
        asset_info: AssetInfoUnchecked::cw20(cw20_incentives_address.clone()),
        allocation_config: None,
    },
]);

let init_msg = InstantiateMsg {
    staking_token: AndrAddr::from_string(cw20_staking_token_address.to_owned()),
    additional_rewards,
    kernel_address: MOCK_KERNEL_CONTRACT.to_string(),
    owner: None,
    modules: Some(vec![Module::new("test", "address", false)]),
};

let staking_addr = app.instantiate_contract(staking_code_id,
↪ Addr::unchecked("owner"), &init_msg, &[], "staking", None).unwrap();

//provide some incentives
let msg = Cw20ExecuteMsg::Transfer { recipient: staking_addr.to_string(),
↪ amount: Uint128::from(10000u128) };
let result = app.execute_contract(Addr::unchecked("owner"),
↪ cw20_incentives_address.clone(), &msg, &[]);
println!("{:?}", result);

//// stake tokens

let msg = Cw20ExecuteMsg::Send {
    contract: staking_addr.to_string(),
    amount: 1u128.into(),
    msg: to_binary(&Cw20HookMsg::StakeTokens {}).unwrap()
};

```



```

    let result = app.execute_contract(Addr::unchecked("user"),
↪ cw20_staking_token_address.clone(), &msg, &[]).unwrap();

    let msg = Cw20ExecuteMsg::Send {
        contract: staking_addr.to_string(),
        amount: 100u128.into(),
        msg: to_binary(&Cw20HookMsg::StakeTokens {}).unwrap()
    };

    let result = app.execute_contract(Addr::unchecked("user2"),
↪ cw20_staking_token_address.clone(), &msg, &[]).unwrap();

    let msg = Cw20ExecuteMsg::Send {
        contract: staking_addr.to_string(),
        amount: 100u128.into(),
        msg: to_binary(&Cw20HookMsg::StakeTokens {}).unwrap()
    };

    let result = app.execute_contract(Addr::unchecked("user3"),
↪ cw20_staking_token_address.clone(), &msg, &[]).unwrap();

    //update index
    let msg = ExecuteMsg::UpdateGlobalIndexes {
        asset_infos:
↪ Some(vec![AssetInfoUnchecked::cw20(cw20_incentives_address.clone())]),
    };

    let result = app.execute_contract(Addr::unchecked("owner"),
↪ staking_addr.clone(), &msg, &[]).unwrap();

    //update time
    update_time(&mut app, 15_780_000/6);

    //update index
    let msg = ExecuteMsg::UpdateGlobalIndexes {
        asset_infos:
↪ Some(vec![AssetInfoUnchecked::cw20(cw20_incentives_address.clone())]),
    };

    let result = app.execute_contract(Addr::unchecked("owner"),
↪ staking_addr.clone(), &msg, &[]).unwrap();

    // Verify pending rewards updated with query.
    let msg = QueryMsg::Staker {
        address: "user".to_string(),
    };

```

```

    let res: StakerResponse = app.wrap().query_wasm_smart(staking_addr.clone(),
↪ &msg).unwrap();
    println!("{:?}",res);

    let msg = QueryMsg::Staker {
        address: "user2".to_string(),
    };

    let res: StakerResponse = app.wrap().query_wasm_smart(staking_addr.clone(),
↪ &msg).unwrap();
    println!("{:?}",res);

    let msg = QueryMsg::Staker {
        address: "user3".to_string(),
    };

    let res: StakerResponse = app.wrap().query_wasm_smart(staking_addr.clone(),
↪ &msg).unwrap();
    println!("{:?}",res);
}

```

## RES-05 Lockdrop Participants Won't Be Able To Withdraw Native Assets

```

#[test]
fn test_exploit(){
    let mut app = App::default();
    let code = ContractWrapper::new(execute_cw20, instantiate_cw20, query_cw20);
    let cw_20_code_id = app.store_code(Box::new(code));

    let cw20_init_msg = cw20_InstantiateMsg{ name: "Token".to_owned(), symbol:
↪ "TOK".to_owned(), decimals: 18u8, initial_balances:
    vec![Cw20Coin {
        amount: 100u128.into(),
        address: "owner".to_string(),
    }
    ], mint: None, marketing: None };

    let cw20_incentives_address= app.instantiate_contract(cw_20_code_id,
↪ Addr::unchecked("owner"), &cw20_init_msg, &[], "Token", None).unwrap();

    let code = ContractWrapper::new(execute, instantiate, query);
    let lockdrop_code_id = app.store_code(Box::new(code));
    let current_timestamp = app.block_info().time.seconds();

    let init_msg = InstantiateMsg {
        // bootstrap_contract: None,
        init_timestamp: current_timestamp,
    }
}

```

```

        deposit_window: DEPOSIT_WINDOW,
        withdrawal_window: WITHDRAWAL_WINDOW,
        incentive_token: cw20_incentives_address.to_string(),
        native_denom: "uusd".to_string(),
        kernel_address: MOCK_KERNEL_CONTRACT.to_string(),
        owner: None,
        modules: None,
    };

    let lockdrop_addr = app.instantiate_contract(lockdrop_code_id,
↪ Addr::unchecked("owner"), &init_msg, &[], "staking", None).unwrap();

    mint_native(&mut app, "user1".to_owned(), "uusd".to_owned(), 500);
    mint_native(&mut app, "user2".to_owned(), "uusd".to_owned(), 500);

    update_time(&mut app, DEPOSIT_WINDOW);

    let msg = ExecuteMsg::DepositNative { };
    let resp = app.execute_contract(Addr::unchecked("user1"), lockdrop_addr.clone(),
↪ &msg, &[coin(500, "uusd")]).unwrap();

    let msg = ExecuteMsg::DepositNative { };
    let resp = app.execute_contract(Addr::unchecked("user2"), lockdrop_addr.clone(),
↪ &msg, &[coin(500, "uusd")]).unwrap();

    let msg = Cw20ExecuteMsg::Send {
        contract: lockdrop_addr.to_string(),
        amount: 100u128.into(),
        msg: to_binary(&Cw20HookMsg::IncreaseIncentives {}).unwrap(),
    };

    let resp = app.execute_contract(Addr::unchecked("owner"),
↪ cw20_incentives_address.clone(), &msg, &[]);

    update_time(&mut app, WITHDRAWAL_WINDOW + 1);

    //enable claims
    let msg = ExecuteMsg::EnableClaims {};
    app.execute_contract(Addr::unchecked("owner"), lockdrop_addr.clone(), &msg,
↪ &[]);

    //claim

```

```

    let msg = ExecuteMsg::ClaimRewards {};
    let result = app.execute_contract(Addr::unchecked("user1"),
↳ lockdrop_addr.clone(), &msg, &[]).unwrap();

    let msg = ExecuteMsg::ClaimRewards {};
    let result = app.execute_contract(Addr::unchecked("user2"),
↳ lockdrop_addr.clone(), &msg, &[]).unwrap();


let balance = query_balance_native(&app, &Addr::unchecked("user1"), "uusd");

let msg = ExecuteMsg::WithdrawNative { amount: None };

let result = app.execute_contract(Addr::unchecked("user1"),
↳ lockdrop_addr.clone(), &msg, &[]).unwrap();
}

```