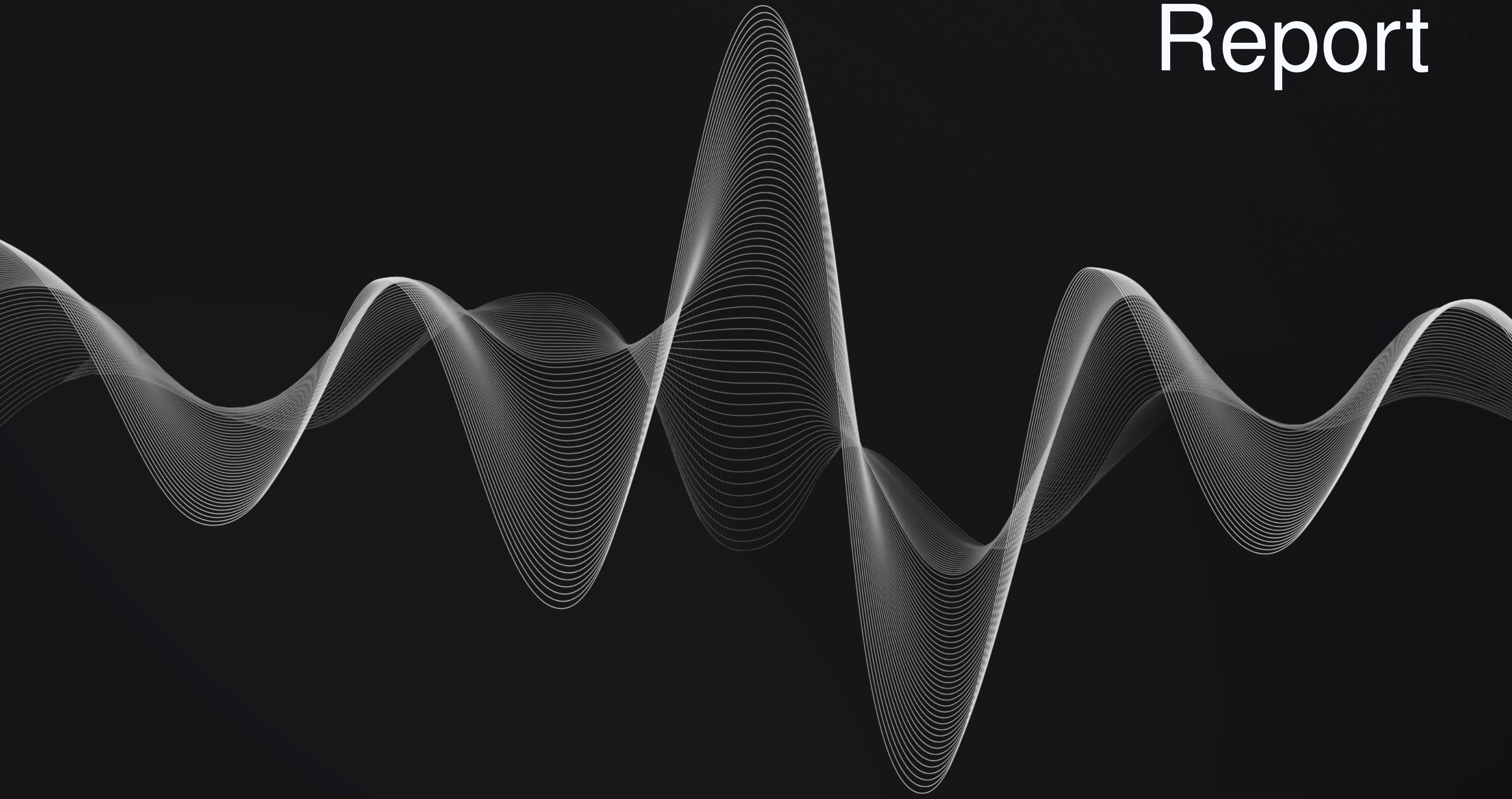
RESONANCE

primex

Primex Finance

Primex Protocol Integrations Audit Report



Blockchain, Emerging Technology, and Web2
CYBERSECURITY PRODUCT & SERVICE ADVISORY

Document Control

PUBLIC **FINAL**(v2.2)

Audit_Report_PRMX-TPI_FINAL_22

Feb 27, 2024	○	v0.1	João Simões: Initial draft
Feb 29, 2024	○	v0.3	Michał Bajor: Added findings
Mar 5, 2024	●	v1.0	Charles Dray: Approved
Mar 29, 2024	○	v1.1	João Simões: Reviewed findings
Jun 3, 2024	●	v2.0	Charles Dray: Finalized
Oct 11, 2024	○	v2.1	João Simões: Reviewed contract updates
Oct 14, 2024	○	v2.2	Charles Dray: Published

Points of Contact	Oleksandr Marukhnenko	Primex Finance	alex@primex.finance
	Charles Dray	Resonance	charles@resonance.security
Testing Team	João Simões	Resonance	joao@resonance.security
	Ilan Abitbol	Resonance	ilan@resonance.security
	Michał Bazyli	Resonance	michal@resonance.security
	Michal Bajor	Resonance	michal.bajor@resonance.security

Copyright and Disclaimer

Contents

Executive Summary

Primex Finance contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between February 20, 2023 and March 5, 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 2 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 10 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 Primex Finance 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.

System Overview

Primex Finance is a decentralized exchange platform powered by smart contract where users can lend and swap tokens, perform trading operations, and earn rewards by utilizing the protocol.

The system is composed of several components: the bucket infrastructure where all the lending and borrowing happens; the trading managers that implement logic to swap, open and close positions, with or without conditions; and the protocol management where Primex admin control emergency pauses, propositions and executions.

As a general workflow, a lender is able to stake collateral on a bucket and receive yield-bearing tokens. The collateral can be used by traders to borrow and open margin positions. The keepers handle the conditions that occur over time, such as, opening limit orders when price as reached the limit price, liquidating users with underwater positions, closing trades with set stop-losses, etc.

By using the different components of the protocol, all actors receive rewards that incentivise further usage. One of such rewards is the PMX token that offers utility and governance capabilities on the protocol.

Additional features were implemented such as new oracle and decentralized exchanges integrations, and modifications to the calculation of protocol fees and keeper rewards were made.

Repository Coverage and Quality

Code	Tests	Documentation
7 / 10	7 / 10	7 / 10

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 and uses 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 (main): `primex-finance/primex-protocol/src/contracts`
- Hash: d8f57547458551ff2ac54abe5d646b9933bd8a24
- Repository: `primex-finance/primex_contracts/src/contracts`
- Hash: eb9933ab438181b0a5bc0c3a72d30a89a10e1ae6

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial-related attack vectors

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 - Solidity EVM

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 Solidity EVM audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Reentrancy attacks
- Frontrunning attacks
- Unsafe external calls
- 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.

RES-01	🟥	Invalid Asset Price Due To Incomplete Price Mappings	▂▃▅▃▂	Resolved
RES-02	🟥	Missing Return Value On _swapExactTokensForTokens() Leads To Incorrect Calculations	▁▁▂▁▁	Resolved
RES-03	🟧	Missing Amount In And Out Calculations For Paraswap Integration	▂▃▅▃▂	Acknowledged
RES-04	🟩	Redundant Code On _withdrawBucketLiquidityFromAave()	▂▃▅▃▂	Resolved
RES-05	🟩	Implementation Not Adhering To the Protocol Specification	▁▁▂▁▁	Acknowledged
RES-06	⬜	Unnecessary Usage Of int256 On minPositionSizeMultiplier	▁▁▂▁▁	Resolved
RES-07	⬜	Lack Of Access Control On setProtocolParamsByAdmin()	▁▁▂▁▁	Acknowledged

Invalid Asset Price Due To Incomplete Price Mappings

High | **RES-PRMX-TPI01** | Business Logic | **Resolved**

Code Section

- `contracts/PriceOracle.sol#191`
- `contracts/PriceOracle.sol#202`

Description

The `PriceOracle` contract is responsible for dictating the prices for given assets. The prices could be fetched using multiple of sources - Uniswap, Pyth and Chainlink to be precise. However, the Pyth and Chainlink sources support only prices denominated in USD. The `PriceOracle` contract keeps the prices in a mapping that is indexed with the asset price and returns that asset's price in USD. Hence, if the requested price is for a pair that does not include USD (for instance BTC/ETH) and Pyth/Chainlink is used as a price source, then the `_getExchangeRate()` function would actually return the price denominated in USD which will lead to invalid calculations.

Recommendation

It is recommended to implement a verification mechanism responsible for assuring that at least one of the assets in the provided pair is USD when using Pyth or Chainlink as a price source.

Status

The issue has been fixed in e65d4db4cb259853e36e188bea59605bbda920f4.

Missing Return Value On _swapExactTokensForTokens() Leads To Incorrect Calculations

High RES-PRMX-TPI02 Business Logic **Resolved**

Code Section

- `contracts/DexAdapter.sol#L461`
- `contracts/DexAdapter.sol#L258-L292`

Description

The functions `performPathsSwap()` and `swapExactTokensForTokens()` make use of the internal function `_swapExactTokensForTokens()` and expect a `uint256[3]` value to be returned. Further calculations are made with this returned value. However, for the `DexType.Paraswap`, no value is returned, both leading to incorrect calculations of values presented to the caller of the functions.

Recommendation

It is recommended to add a return statement to properly perform the necessary calculations.

Status

The issue has been fixed in e65d4db4cb259853e36e188bea59605bbda920f4.

Missing Amount In And Out Calculations For Paraswap Integration

Medium | **RES-PRMX-TPI03** | Business Logic | **Acknowledged**

Code Section

- `contracts/DexAdapter.sol#L207-L225`
- `contracts/DexAdapter.sol#L347-L364`
- `contracts/DexAdapter.sol#L420-L437`

Description

The functions `getAmountsIn()`, `getAmountsOut()`, and `getGas()` do not implement the possibility of calculating beforehand the amount of tokens and gas that will be used to swap through the function `_swapExactTokensForTokens()`. Users will have to perform swaps by trial and error without previously knowing accurate calculation estimates.

Recommendation

It is recommended to implement similar functionality on `getAmountsIn()`, `getAmountsOut()`, and `getGas()` to the ones already implemented for other decentralized exchanges integrations.

Status

The issue was acknowledged by Primex's team. The development team stated "Paraswap itself doesn't have any methods to get amounts in/out on-chain, so users should use the results from Paraswap API. Regarding the getGas() method, it was used for the on-chain splitter that we don't support for new DEXs, so it's normal that it doesn't support Paraswap".

Redundant Code On _withdrawBucketLiquidityFromAave()

Low

RES-PRMX-TPI04 Code Quality **Resolved**

Code Section

- `contracts/Bucket/Bucket.sol#L679-L697`
- `contracts/Bucket/BucketExtension.sol#L99-L117`

Description

The function `_withdrawBucketLiquidityFromAave()` is duplicated on both `Bucket` and `BucketExtension` contracts. This fact makes the code more complex and less readable, while also wasting unnecessarily more bytecode on the blockchain.

Recommendation

It is recommended to merge both functions into one and change their calls as necessary.

Status

The issue has been fixed in e65d4db4cb259853e36e188bea59605bbda920f4.

Implementation Not Adhering To the Protocol Specification

Low

RES-PRMX-TPI05 Data Validation **Acknowledged**

Code Section

- `contracts/PriceOracle/PriceOracle.sol#L170-L176`

Description

Provided protocol specification, and specifically the recent changes, regarding Oracle contract's implementation state that if Oracle Route's length is equal to four, then Pyth and Chainlink Oracles cannot be neither the first nor the last step in the whole path. This requirement was formulated to assure only expected token conversions are permitted. However, it was observed that the implementation of such mechanism is not complete. Namely, it does verify that Pyth and Chainlink Oracles are not the first step in the four-step route, but they do not assert they are not the last. Hence, the scenarios that were supposed to be prevented by design are actually possible.

Recommendation

It is recommended to add another `require` statement to assert that neither Pyth nor Chainlink Oracles are the last step of the four-step Oracle route.

Status

The issue was acknowledged by Primex's team. The development team stated "Regarding the check of the last oracle can't be Pyth, it's ok if a user can calculate a price of an asset to USD as a view, what is important as such a price can't be used inside a transaction.".

Unnecessary Usage Of int256 On minPositionSize-Multiplier

Info **RES-PRMX-TPI06** Code Quality **Resolved**

Code Section

- `contracts/KeeperRewardDistributor/KeeperRewardDistributor.sol#L213-L222`

Description

The contract `KeeperRewardDistributorStorageV2` implements the state variable `minPositionSizeMultiplier` as an `int256`. The usage of this state variable throughout the code assumes that it will always contain an unsigned integer. Therefore, various checks may be removed to optimize gas.

Recommendation

It is recommended to use unsigned integer variables whenever the value being stored is never supposed to be negative.

Status

The issue has been fixed in e65d4db4cb259853e36e188bea59605bbda920f4.

Lack Of Access Control On setProtocolParams-ByAdmin()

Info **RES-PRMX-TPI07** Access Control **Acknowledged**

Code Section

- `contracts/PositionManager/PositionManager.sol#L79-L81`

Description

The function `setProtocolParamsByAdmin()` does not implement access control allowing any user to delegate calls to `PositionManagerExtension`.

While this does not present an immediate threat since all administrative functions on `PositionManagerExtension` possess proper access control, current or future functions that do not have any access control implemented may be at risk.

Recommendation

It is recommended to implement proper access control on the function to prevent normal users from calling administrative functions.

Status

The issue was acknowledged by Primex's team. The development team stated "The access to administrative functions in PositionManager is controlled at the implementation level. The method setProtocolParamsByAdmin is utilized by various types of admins – currently, MEDIUM and BIG – and we would have to verify msg.sender twice. This will increase gas expenditure. If we create two separate methods for different types of admins, we will be required to define the list of permitted methods for each admin type. This approach increases the bytecode size and creates new space for errors. Therefore, we adhere to the principle that all access control checks for administrative methods must occur within the implementation itself.".

Proof of Concepts

No Proof-of-Concept was deemed relevant to describe findings in this engagement.