ASTRARIZON

ASTRARIZON LABS

Code Review Report

EXECUTIVE SUMMARY

1. AUDIT OVERVIEW

This document outlines the security audit conducted for the smart contracts contained within the "Photon Crosschain Messaging MultiversX" repository, specifically the "endpoint-contract" and the "oracle-price-spotter." Integral to enhancing blockchain interoperability and communication, these contracts support secure and efficient cross-chain interactions essential for data sharing and transactions across various blockchain platforms, with a particular focus on the MultiversX ecosystem.

The primary objective of this audit is to methodically examine the codebase of each smart contract to uncover potential security vulnerabilities and recommend areas for improvement, adhering to the best practices in smart contract development. This effort aims to bolster the contracts' security and operational efficiency.

Key areas covered in the audit include:

- The examination of smart contracts' architectures and coding patterns to identify security vulnerabilities or areas that could be optimized.
- The analysis of specific functionalities, such as cross-chain messaging in the "endpoint-contract" and price spotting in the "oracle-price-spotter," with a focus on authentication, validation, error handling, and secure operations.
- The evaluation of compliance with recognized smart contract development standards, especially in terms of security, upgradability, and interoperability.
- Extensive functional testing under various scenarios to detect operational vulnerabilities.

The purpose of this security assessment is to deliver detailed insights and recommendations for the enhancement of the "endpoint-contract" and "oracle-price-spotter" smart contracts. It is acknowledged that identifying all potential security issues is a complex endeavor, given the evolving landscape of blockchain technology. Therefore, this audit is committed to contributing towards the continuous improvement of the contracts' security framework.

2. OBJECTIVE AND SCOPE

2.1. OBJECTIVE

The objective of this security audit is to conduct a comprehensive analysis of the ”Photon Crosschain Messaging MultiversX” smart contracts, designated as ”endpoint-contract” and ”oracle-price-spotter.” The audit aims to critically assess the security posture, identify vulnerabilities, ensure code integrity, and evaluate adherence to the highest standards of smart contract development. By providing this scrutiny, the audit is intended to uncover potential security flaws, suggest robust mitigation strategies, and support the enhancement of the contracts’ security and reliability.

2.2. SCOPE

The scope of the audit is detailed as follows:

- **Code Review:** A line-by-line inspection of the smart contracts’ source code will be carried out to uncover security weaknesses, code smells, and anti-patterns that could compromise the contract’s integrity.
- **Functionality Analysis:** The contracts’ functionalities will be tested to confirm that they perform as expected across a variety of scenarios, including edge cases and failure modes.
- **Standards Compliance:** The code will be measured against established blockchain and smart contract development standards, including but not limited to the General Security Guidelines for Ethereum and the Smart Contract Weakness Classification Registry.
- **Client Specifications:** Verification that the smart contracts fulfill the client’s technical specifications and operational requirements will be conducted meticulously.
- **Attack Vector Assessment:** The contracts will be evaluated to estimate their resilience against known and hypothetical attack vectors prevalent in the industry.
- **Contract Logic Verification:** Ensuring that the contract logic is sound and does not contain errors that could lead to unintended behaviors or exploitation.
- **Gas Usage and Optimization:** An analysis will be done to determine the efficiency of gas usage by the contracts and provide recommendations for optimization where possible.
- **Dependency Analysis:** Any external dependencies and libraries used by the smart contracts will be analyzed for security and stability.

The audit process will conclude with a detailed report comprising the identified issues, their potential impact on the contract’s security and functionality, and a set of recommendations for addressing each identified concern. It is important to note that the audit is based on the state of the smart contract code at the time of the audit and within the scope defined above. As the security landscape evolves and new vulnerabilities are discovered, ongoing reviews and updates to the smart contract may be required to maintain a robust security posture.

The audit findings should be integrated into a comprehensive security strategy. However, this audit does not replace internal security reviews or formal contract logic verification and does not guarantee absolute security.

3. REPOSITORY AND COMMIT REFERENCE

For the purpose of this security audit, the ”Photon Crosschain Messaging MultiversX” smart contracts, namely ”endpoint-contract” and ”oracle-price-spotter,” were reviewed. It is important to note that the repository containing the smart contracts is private, and access is restricted to authorized individuals. This ensures the confidentiality of the codebase while still allowing for a comprehensive security assessment by the auditing team.

Repository Access: The repository is hosted on a private GitHub account managed by the Entangle Protocol team, making it inaccessible to the public. Access to the repository was granted to the auditing team on a temporary basis and solely for the purposes of this security assessment.

Commit Reference: The audit was based on a specific snapshot of the code identified by the commit hash. This commit serves as the reference point for the audit, providing a static codebase for a consistent and repeatable analysis. For confidentiality and security reasons, the commit hash is not disclosed in this document.

Codebase Stability: The code at the commit hash mentioned above was reviewed without any subsequent modifications. The stability of the codebase at this commit is critical for the validity of the audit findings. Any changes made to the code after the referenced commit may not be covered by this audit and could potentially alter the effectiveness of the recommendations provided herein.

Future Audits: Should there be any updates or modifications to the smart contracts after the date of the referenced commit, it is recommended that an additional audit be conducted to ensure that new or altered code maintains a high standard of security and does not introduce new vulnerabilities.

In accordance with best practices and to maintain the integrity of the audit process, detailed findings and associated recommendations will reference specific functions, lines of code, or code structures without revealing proprietary information or sensitive details of the implementation.

4. CONTRACTS UNDER AUDIT

The security audit thoroughly examines two principal smart contracts from the ”Photon Crosschain Messaging MultiversX” system: the ”endpoint-contract” and the ”oracle-price-spotter.” Each contract encapsulates specific functionalities that are critical to the operation of the crosschain messaging framework.

4.1. ENDPOINT-CONTRACT

The ”endpoint-contract” functions as the core interface for executing crosschain operations and governance processes. It is structured to handle various aspects of crosschain messaging, including:

- Initialization of governance parameters and access control mechanisms.
- Validation and execution of crosschain operations based on consensus verification.
- Proposal of new operations and governance actions.
- Secure storage of executed operations and protocol configurations.
- Administration of protocol information and governance through a set of predefined operations.

This contract’s audit will focus on the secure implementation of its crosschain functionalities, proper access controls, and efficient governance operations.

4.2. ORACLE-PRICE-SPOTTER

The ”oracle-price-spotter” contract is designed to capture and relay asset price information accurately within the ecosystem. It is responsible for:

- Managing access controls for administrative functions and price updates.
- Setting and updating global and onchain prices for various assets.
- Ensuring that price data is current and preventing the recording of obsolete or outdated price information.
- Storing asset price ticks with associated timestamps for historical tracking.

The audit will assess the contract’s mechanisms for price data handling, update authentication, and the integrity of its storage patterns.

The audit aims to verify that both contracts function cohesively within the Photon Crosschain Messaging system to achieve their intended outcomes while maintaining a high standard of security. Emphasis will be placed on the analysis of smart contract permissions, data validation processes, error handling, event management, and the appropriate use of the MultiversX framework features. The assessment will be performed with the goal of identifying any potential vulnerabilities that could affect the reliability and security of these contracts.

In both cases, the audit will respect the privacy of the code by not disclosing proprietary logic or sensitive details. Instead, it will refer to the components’ structure and features in a way that provides transparency about the audit process while preserving confidentiality.

5. SEVERITY DISTRIBUTION

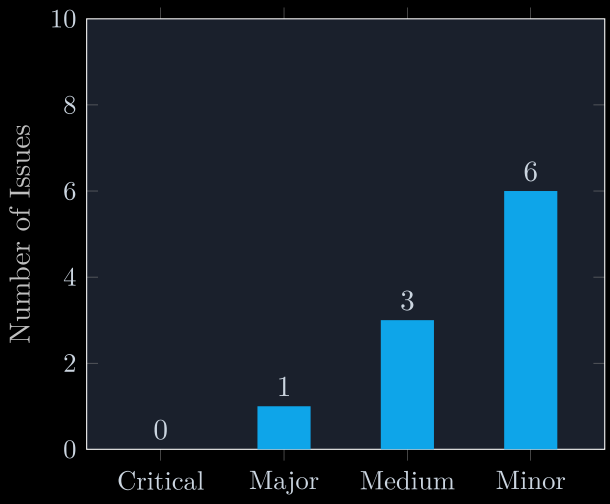
This report presents an analysis focusing on the vulnerability severity distribution within the ”Photon Crosschain Messaging MultiversX” smart contracts, specifically the ”Endpoint-Contract” and ”Oracle Spotter Contract.” Vulnerabilities are classified into four categories based on their potential impact on system security and operations: Critical, Major, Medium, and Minor. The classification criteria include exploitability, impact on system functionality, and the complexity of mitigation measures.

SEVERITY LEVELS DEFINED

- **Critical**: Vulnerabilities posing immediate and severe risks to system or data integrity, requiring urgent remediation.
- **Major**: Significant risks that could lead to substantial security breaches or loss of functionality.
- **Medium**: Risks affecting system performance or security to a lesser degree, warranting timely attention.
- **Minor**: Issues of low risk, mainly concerning optimization and code cleanliness, with minimal direct impact on system security.

ENDPOINT-CONTRACT VULNERABILITY DISTRIBUTION

The Endpoint-Contract is integral to the functionality of crosschain operations and governance within the ecosystem. Below is a graphical representation of the vulnerabilities identified across different severity levels:

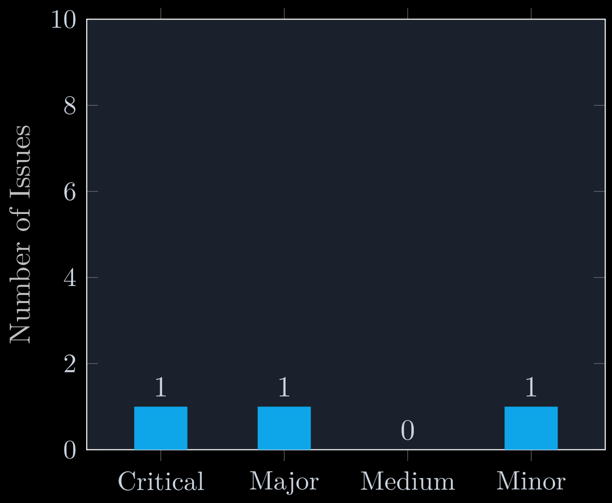


Distribution of identified vulnerabilities in the Aggregation Spotter Contract.

In summary, the vulnerability analysis of the Endpoint-Contract revealed a total of 10 issues, segmented into 0 Critical, 1 Major, 3 Medium, and 6 Minor vulnerabilities. This distribution highlights the absence of critical-level vulnerabilities while identifying areas for improvement primarily at the minor and medium severity levels, emphasizing the contract’s robustness with room for refinement in specific areas to enhance security and operational efficiency.

ORACLE SPOTTER CONTRACT VULNERABILITY DISTRIBUTION

The Oracle Spotter Contract plays a pivotal role in ensuring the accuracy of asset price data within the ecosystem. The distribution of identified vulnerabilities across different severity levels is illustrated in the figure below:



Distribution of identified vulnerabilities in the Oracle-Price Spotter Contract.

Upon review, the Oracle Spotter Contract's vulnerability analysis revealed a total of 3 issues, classified as 1 Critical, 1 Major, 0 Medium, and 1 Minor vulnerabilities. This assessment underscores a significant emphasis on addressing severe threats, with a critical issue having been identified and subsequently fixed. The presence of a major issue, also fixed, alongside a couple of minor improvements points towards a proactive approach to securing the contract against potential vulnerabilities. The absence of medium-level vulnerabilities further signifies targeted prioritization towards the most impactful security concerns.

6. SUMMARY OF FINDINGS

The security audit conducted on the ”Photon Crosschain Messaging MultiversX” smart contracts—specifically, the ”Endpoint-Contract” and the ”Oracle Spotter Contract”—identified a series of vulnerabilities and areas for optimization. These findings span across various aspects of smart contract development, including access control, data validation, and operational efficiency. Key observations and recommendations are summarized as follows:

1. **Access Control and Initialization:** A significant observation was the absence of a mechanism to add executors during the initialization of new protocols in the Endpoint-Contract. This limitation restricts the contract’s governance capabilities. It’s recommended to enforce at least one executor for the effective operation of newly added protocols.
2. **Data Validation:** The audit pinpointed a lack of validation for array lengths and consensus target rates. Specifically, the unverified lengths of allowed keepers and executors could lead to governance lockout, and an unbounded consensus target rate might destabilize protocol operations. Introducing validation checks is advised to mitigate these risks.
3. **Operational Efficiency:** The representation of the consensus target rate and timestamp data types were found to be non-optimal, suggesting a change to `u64` for better performance.
4. **Code Optimization:** Recommendations include removing unused variables and redundant logic, such as the allowed proposers logic, to streamline the codebase and improve contract efficiency.
5. **Fee Collection Mechanism:** The current approach to fee collection lacks a payable endpoint, complicating the process. Establishing a clear mechanism for fee transactions is essential for operational clarity.
6. **Minor Observations:** Typos, variable naming inconsistencies, and the presence of test code in the production environment were noted. Addressing these points will enhance code readability and professionalism.
7. **Resolved Findings:** Critical and major issues identified in previous audits, such as unsafe handling of `unwrap()` and inefficient timestamp management, have been addressed. This proactive approach underscores the commitment to maintaining high security standards.

> **All identified vulnerabilities have been addressed and resolved**, significantly strengthening the contracts’ security framework and operational integrity. This resolution process reflects a steadfast dedication to upholding the highest standards of security and ensuring the robustness and reliability of the Photon Crosschain Messaging MultiversX smart contracts.

METHODOLOGY

7. AUDIT PROCESS

The security audit for the ”Photon Crosschain Messaging MultiversX” smart contracts is executed with a focus on manual examination and testing methods. This comprehensive and detailed approach ensures the identification and remediation of potential vulnerabilities, emphasizing manual scrutiny throughout various phases of the audit.

PREPARATION AND PLANNING

This foundational phase involves an exhaustive review of the system’s architecture, particularly the ”Endpoint-Contract” and ”Oracle Spotter Contract.” The audit team prepares by understanding the operational logic, functionalities, and intended security measures of these contracts. Clear objectives are set, and a tailored audit plan is developed to guide the thorough examination process.

STATIC ANALYSIS

Static analysis is the first step in the active audit process, where the source code of the contracts is meticulously reviewed by experts. This phase aims to detect vulnerabilities such as logic errors, permission issues, and security flaws through manual inspection. The emphasis on manual review allows for a nuanced understanding of the codebase, enabling the identification of subtle issues that might not be apparent without deep domain knowledge.

DYNAMIC ANALYSIS

Building on the insights gained from static analysis, dynamic analysis involves manual testing of the contracts under simulated conditions. This phase tests the contracts’ responses to various inputs and scenarios, looking for vulnerabilities that manifest only during execution. The focus is on hands-on interaction with the contracts, simulating attacks, and testing edge cases to ensure comprehensive vulnerability identification.

SECURITY BEST PRACTICES REVIEW

The contracts are also reviewed against established smart contract development and security best practices. This manual evaluation identifies areas where the contracts diverge from industry standards or could benefit from security enhancements, offering a path to bolster the contracts’ defenses.

VULNERABILITY ASSESSMENT AND SEVERITY CLASSIFICATION

All findings from the manual static and dynamic analyses and the best practices review are collated into a vulnerability assessment report. Each issue is manually classified by its severity, informed by its potential impact on the system and the complexity of its exploitation. This classification aids in prioritizing remediation efforts according to the risk posed.

REPORTING AND RECOMMENDATIONS

A detailed report concludes the audit process, listing all identified vulnerabilities along with their severity levels and providing specific, actionable recommendations for each issue. This manual compilation serves as a comprehensive guide for the development team to systematically address and resolve identified security concerns.

REMEDIATION VERIFICATION

Once remediation efforts are undertaken, a final manual verification step confirms the effective resolution of all vulnerabilities. This ensures that the fixes are properly implemented and that the contracts meet stringent security standards post-audit.

DETAILED FINDINGS

8. ENDPOINT CONTRACT

8.1. CRITICAL VULNERABILITIES

> This audit has not uncovered any critical vulnerabilities within the smart contract, indicating a robust level of security against threats that could lead to severe implications such as asset theft or system compromise.

8.2. MAJOR VULNERABILITIES

8.2.1. Unsafe Handling of ‘unwrap()‘ Method Calls

Observation

The smart contract code employs ‘unwrap()‘ method calls across several points. This usage poses significant risk, as invoking ‘unwrap()‘ on a ‘None‘ value, which might occur during failed decoding processes, triggers a panic within the contract.

Recommendation

Implement pattern matching for error handling to replace ‘unwrap()‘ calls, thereby enhancing the contract’s resilience to erroneous states and preventing unexpected panics.

Implementation Suggestion

```
let call_result: Result<SetPrice<Self::Api>, _> = codex::top_decode_from_nested
    (photon_msg.params);
let call = match call_result {
    Ok(decoded_call) => decoded_call,
    Err(_) => return,
};
```

8.3. MEDIUM VULNERABILITIES

8.3.1. Potential for Governance Protocol Lockout

Observation

The ‘init_gov()‘ function lacks checks for the length of ‘allowed_keepers‘ and ‘executors‘ arrays. This oversight may result in a scenario where governance operations are rendered unexecutable, leading to a complete protocol lockout.

Recommendation

Incorporate length verification within the ‘init_gov‘ function for ‘allowed_keepers‘ and ‘executors‘ to ensure these arrays are properly initialized, mitigating the risk of a governance lockout.

Implementation Suggestion

```
require!(!executors.is_empty(), "At least one executor required!");
require!(!allowed_keepers.is_empty(), "At least one keeper required!");
```

8.3.2. Consensus Target Rate Validation Absence

Observation

The contract does not enforce validation for the ‘consensus_target_rate‘ setting, potentially allowing values that are excessively high or low, which could destabilize the protocol or impair its operational integrity.

Recommendation

Implement validation checks to ensure ‘consensus_target_rate‘ is within predefined sensible limits, safeguarding the protocol’s functionality and stability.

Implementation Suggestion

```

fn check_valid_consensus_rate(&self, consensus_target_rate: u64) {
    require!(consensus_target_rate > 0, " Rate too low.");
    require!(consensus_target_rate <= RATE_DECIMALS," Rate too high.");
}
```

8.3.3. Non-Payable Endpoint for Protocol Fee Collection

Observation

The mechanism for collecting protocol fees lacks a designated payable endpoint for ‘ngl_tokens‘, complicating the fee transfer process to the ‘fee_collector‘. Additionally, there is no verification mechanism to ensure sufficient ‘ngl_tokens‘ are available to cover the fees.

Recommendation

- Establish a designated payable endpoint within the contract for ‘ngl_tokens‘ deposits, clarifying the fee payment process.

- Implement checks to verify the presence of adequate ‘ngl_tokens‘ before proceeding with fee transactions, preventing failures due to insufficient funds.

Implementation Suggestion

```
#[endpoint(payable_endpoint)]
#[payable("*")]
fn payable_endpoint(
    &self,
    #[payment_token] token: EgldOrEsdtTokenIdentifier,
    #[payment_nonce] nonce: u64,
    #[payment_amount] amount: BigUint,
) {
    require!(token == self.stored_payable_token_identifier().get(), "Invalid
        token used!");

    //before paying fees
    require!(amount >= MINIMUM_TOKENS_REQUIRED, "Not enough tokens!");
}
```

8.4. MINOR VULNERABILITIES

8.4.1. Inefficient Nonce Management

Observation

The practice of storing nonces as ‘BigUint‘ is identified as unnecessary, given that nonces are managed as ‘u64‘ within the MultiversX SDK, suggesting an area for optimization.

Recommendation

Adopt ‘u64‘ for nonce storage, aligning with the SDK’s handling and improving the contract’s efficiency.

Implementation Suggestion

```
 fn nonce_function(&self,..., nonce: u64) {
 // ...
 }
```

8.4.2. Hardcoded Function Selectors

Observation

The use of hardcoded hexadecimal values for function selectors in ‘handle_gov_operations()‘ diminishes code readability and maintainability.

Recommendation

- Introduce an enumeration (‘enum‘) to represent function selectors, enhancing code clarity and reducing error susceptibility during updates.
- Replace direct hexadecimal references with enum variants, fostering readability and maintainability.

Implementation Suggestion

```rust
pub enum GovFunctionSelector {
    SomeProtocol = 0x1234abcd,
    AnotherProtocol = 0x2345bcde,
    // ...
}

fn handle_gov_operation(&self, function_selector: [u8; 4], ...) {
    let selector = GovFunctionSelector::from_be_bytes(function_selector);
    match selector {
        GovFunctionSelector::SomeProtocol => { ... }
        GovFunctionSelector::AnotherProtocol => { ... }

        _ => require!(false, "Unknown Selector"),
    }
}
```

8.4.3. Suboptimal Representation of Consensus Target Rate

Observation

The current representation of ‘consensus_target_rate‘ as a type other than ‘u64‘ introduces unnecessary complexity and operation cost.

Recommendation

Modify the data type of ‘consensus_target_rate‘ to ‘u64‘, optimizing operation efficiency and simplifying calculations.

Implementation Suggestion

```rust
fn some_target_rate_funct(&self,..., consensus_target_rate: u64) {
// ...
}
```

8.4.4. Typographical Errors in Variable Naming

Observation

Variable naming inconsistencies and typographical errors, such as ‘uniquie_signers‘ instead of ‘unique_signers‘, detract from code clarity and professionalism.

Recommendation

Correct all typographical errors in variable names, adhering to consistent naming conventions to enhance readability and accuracy.

8.4.5. Inefficient Representation of Byte Arrays

Observation

Certain data structures are represented as byte arrays ‘[u8; 4]‘ where a simpler ‘u32‘ representation would suffice, indicating an area for optimization.

Recommendation

Transition byte array representations to ‘u32‘ where applicable, streamlining data handling and improving code efficiency.

8.4.6. Inconsistent Naming Conventions

Observation

A mix of camelCase and snake_case is used for storage variable naming, leading to inconsistencies within the codebase.

Recommendation

Standardize the naming convention for storage variables, preferably adopting snake_case, to unify code style and improve readability.

Implementation Suggestion

```
// convert this
#[derive(StorageMapper)]
#[storage_mapper("someStorageMapper")]

// into this
#[derive(StorageMapper)]
#[storage_mapper("some_storage_mapper")]
```

9. ORACLE-SPOTTERS CONTRACT

9.1. CRITICAL VULNERABILITIES

9.1.1. Lack of Input Validation and Access Control in Price Setting

Observation

A critical vulnerability has been identified within the ‘photon_call‘ endpoint, characterized by an absence of input validation and access control mechanisms. This deficiency allows any entity, potentially including malicious actors, to manipulate prices unchecked, posing a significant risk to the integrity of the system.

Recommendation

- Implement a ‘require‘ statement to restrict the execution of this endpoint exclusively to the designated ‘spotter‘ role.
- Enforce a ‘require‘ statement to validate that the provided timestamp is not set in the past, thereby averting retroactive price manipulations.

Implementation Suggestion

```
require!(self.spotter().get() == self.blockchain().get_caller(), "Unauthorized:
     Not Spotter");
require!(timestamp >= self.blockchain().get_block_timestamp(), "Invalid:
    Timestamp in the past");
```

9.2. MAJOR VULNERABILITIES

9.2.1. Risks Associated with Unchecked ‘unwrap()‘ Calls

Observation

The contract code makes extensive use of ‘unwrap()‘ calls, a practice that harbors significant risk. Specifically, invoking ‘unwrap()‘ on a ‘None‘ value, such as what might occur following a decoding failure, triggers an immediate contract panic.

Recommendation

Transition from ‘unwrap()‘ to pattern matching for error handling, thus enhancing the robustness of the contract by gracefully managing potential error states.

Implementation Suggestion

```rust
let call_result: Result<SetPrice<Self::Api>, _> = codex::top_decode_from_nested
    (photon_msg.params);
let call = match call_result {
    Ok(decoded_call) => decoded_call,
    Err(_) => return,
};
```

9.3. MEDIUM VULNERABILITIES

> The audit process did not uncover any vulnerabilities of medium severity within the smart contract, indicating a satisfactory level of protection against intermediate threats.

9.4. MINOR VULNERABILITIES

9.4.1. Inefficient Management of Timestamps

Observation

The existing approach to timestamp management, which utilizes ‘BigUint‘, is identified as unnecessarily complex. Timestamp values are well within the range of a ‘u64‘ type, suggesting an area for optimization.

Recommendation

Adopt ‘u64‘ for timestamp storage, aligning with best practices for simplicity and efficiency in resource usage.

Implementation Suggestion

```rust
 fn some_timestamp_function(&self,..., timestamp: u64) {
 // ...
 }
```

9.4.2. Improvements in Code Organization

Observation

The consolidation of structures, traits, and tests within a single file detracts from the code's readability and maintainability.

Recommendation

Reorganize the codebase by segregating struct definitions, traits, tests, and functional code into distinct files. This reorganization should be accompanied by thoughtful naming conventions to enhance clarity and facilitate easier navigation through the code.

CONCLUSION

10. FINAL ASSESSMENT

AUDIT SUMMARY

In this final assessment, we summarize the outcomes of our security audit of the ”Photon Crosschain Messaging MultiversX” smart contracts: ”endpoint-contract” and ”oracle-price-spotter.” Our comprehensive examination focused on identifying and evaluating vulnerabilities across various severity levels. We are pleased to confirm that all identified vulnerabilities have been effectively addressed.

SECURITY ENHANCEMENT

The remediation efforts undertaken have significantly improved the security profiles of both contracts, underscoring a strong commitment to maintaining high security standards within the MultiversX ecosystem. Despite this progress, it is important to recognize the dynamic nature of blockchain security, necessitating ongoing vigilance.

RECOMMENDATIONS AND FUTURE OUTLOOK

Regular audits, continuous monitoring, and updates remain essential to mitigate against new and emerging security threats. While this audit offers an in-depth view of the current security status, it is not a definitive guarantee against future vulnerabilities. Blockchain technology and smart contract development are rapidly evolving fields, requiring a proactive and adaptive security approach.

CONCLUDING REMARKS

The proactive measures implemented in response to the audit findings have positively influenced the overall security and functionality of the Photon Crosschain Messaging system. We strongly recommend continuing these practices of regular auditing and timely updates as part of a comprehensive security strategy. This commitment is crucial for ensuring the long-term security and resilience of the system in the evolving web3 landscape.

DISCLAIMER

This security audit report is presented by Astrarizon and is intended solely for informational purposes. Astrarizon makes no representations or warranties, either expressed or implied, about the security, accuracy, reliability, or completeness of the information contained in this report or the security of the code audited.

The findings of this report are based on the state of the code at the time of the audit and should not be interpreted as a guarantee of the code's security or functionality. It is important to note that the security landscape is continually evolving, and new vulnerabilities may emerge over time. Therefore, this report should not be seen as a comprehensive or final assessment of the code's security status.

Astrarizon expressly disclaims any liability for any direct, indirect, incidental, consequential, or punitive damages arising out of or in any way connected with the access to, use of, or reliance on this report. This includes any liability arising from the report's findings, recommendations, or any errors or omissions therein.

This report does not constitute legal, financial, or investment advice. Any action taken upon the information in this report is strictly at your own risk. Users are advised to conduct their own due diligence and consult with appropriate professional advisors before making any decisions based on this report.

By using or relying on this report, you agree to indemnify and hold harmless Astrarizon and its affiliates, employees, agents, and representatives from any claims, damages, losses, liabilities, costs, or expenses (including reasonable attorneys' fees) arising from such use or reliance.

This disclaimer forms an integral part of the security audit report and should be considered as such.