RATHER LABS

Blockchain Technical Partners

Security Review Report

DUST.FUN Protocol

#rl001

January 2025

Security Review Report

Index

1. Report Summary

This document details the security review conducted by Rather Labs for **Dust.Fun Protocol**. The scope of the audit encompassed a thorough review of two smart contracts, `EvmDustTokens` and `UniversalDApp`, comprising a total of 744 lines of Solidity code, as evaluated in commit hash 7266ca919be54c6976918349c8c435f637c7a6ec,

The audit was carried out using a multi-faceted approach, including the following methodologies:

- **Manual Code Analysis**: To identify logical vulnerabilities, assess code quality, and ensure adherence to best practices.
- **Automated Tools**: Leveraging static analysis and symbolic execution to detect potential security vulnerabilities.

This report identifies and classifies 6 findings as 1 Medium, 3 Low and 2 Informational comments.

A summary of these findings is presented in Section 6. Summary of Finding

Report Structure

The report is organized as follows:

- **Section 2**: Provides a summary of the protocol, including technical infrastructure and technology stack.
- **Section 3**: Lists the audited files within the scope of the review.
- **Section 4**: Presents an overview of the protocol's functionality.
- **Section 5**: Explains the audit methodology to assess the risks.
- **Section 6**: Summarizes the identified findings and their classifications.
- **Section 7**: Details each finding, including its severity, potential impact, and recommended actions.
- **Section 8**: Evaluates the documentation provided by the Dust.Fun team for this audit.
- **Section 9**: Reviews the test suite, including the results of compilation and test outputs.
- **Section 10**: Summarizes the audit findings and provides final recommendations.
- **Section 11**: Introduces Rather Labs and its approach to security assessments.
- **Section 12**: Concludes with a disclaimer outlining the scope and limitations of this report.

2. Protocol Summary

The **Dust.Fun** protocol is designed to simplify the management of fragmented token balances left in wallets and maximize their value. Address the concept of "dust" tokens as small balances of tokens, often unused, that remain hosted between wallets on different blockchains. The protocol enables users to consolidate these scattered balances by converting them into a single, higher-value token of their choice on a target blockchain.

The protocol is deployed across supported EVM-compatible blockchains and Bitlayer, with integration into ZetaChain to facilitate cross-chain communication and interoperability. This allows users to aggregate tokens from multiple source chains and swap them into a chosen token on the destination chain through a single transaction. The platform's primary goal is to enhance efficiency in wallet management and recover value from otherwise neglected assets, empowering users to transform scattered token balances into meaningful holdings on any supported chai

2.1. Technical Infrastructure

The protocol relies on ZetaChain's decentralized cross-chain messaging system to enable secure and seamless interaction between source, intermediary (ZetaChain), and destination chains.
The infrastructure consists of the following key components:

1. **EvmDustTokens Contract**: Deployed on source and target chains, it handles ERC-20 token management, swaps, and cross-chain bridging.
2. **UniversalDapp Contract**: Deployed on ZetaChain, it receives tokens from the source chain, facilitates swaps, and manages cross-chain communication.
3. **ZetaChain Infrastructure**:
    - **Gateway Contract**: Handles cross-chain token transfers and messaging between source/destination chains.
    - **Oracles/Relayers**: Responsible for receiving and triggering appropriate contract calls on different chains.

2.2 Technology Stack

The protocol leverages the following technology stack:

1. **Smart Contracts**:
    - **Solidity**: Audited smart contracts are written in Solidity (version 0.8.19), adhering to modern standards for security and performance.
    - **OpenZeppelin Libraries**: Utilized for access control, safe transfers, and other functionalities to enhance security.
2. **Cross-Chain Infrastructure**:

    - **ZetaChain**: A Layer-1 blockchain supporting native cross-chain interoperability and providing the protocol with relayers and gateways for seamless token transfers.
3. **DEX Integration**:
    - **Uniswap**: For token swapping on source and destination chains.
4. **Standards**:
    - **ERC-20**: For token compliance.
    - **Permit2 (Uniswap)**: Enables signature-based, gas-efficient token approvals for batch operations.
5. **Oracles/Relayers**:
    - ZetaChain oracles are used to validate cross-chain communication and ensure data integrity during token transfers and swaps.

3. Audited Files

#	Files	Total lines	Line of Code	Comments	Blank lines	Comments ratio
1	packages/foundry/contracts/ **EvmDustTokens.sol**	529	352	118	59	33.52%
2	packages/foundry/contracts/ **UniversalDApp.sol**	215	128	57	30	44.53%
	TOTAL	**744**	**480**	**175**	**89**	**36.45%**

4. Protocol Overview

Protocol Components

The **Dust.Fun protocol** is a decentralized system designed to consolidate fragmented ERC-20 token balances ("dust") across wallets and chains into a single, higher-value token. It operates across EVM-compatible blockchains and leverages ZetaChain for cross-chain interoperability.

The primary components of the system are:

1. **EvmDustTokens Contract**:
    - **Location**: Deployed on supported EVM-compatible blockchains.
    - **Responsibilities**:
        - Facilitates user interactions, such as initiating token swaps and bridging operations.
        - Manages ERC-20 token operations, including transferring tokens to ZetaChain via its gateway.
    - **Capabilities**:
        - Executes swaps locally on the source chain via integrated DEXs (e.g., Uniswap).
        - Encodes the cross-chain payload to initiate transactions on the destination chain.
2. **UniversalDApp Contract**:
    - **Location**: Deployed on ZetaChain.
    - **Responsibilities**:
        - Processes cross-chain messages sent by the source chain's gateway.
        - Handles token swaps, bridging, and withdrawals on ZetaChain.
    - **Capabilities**:
        - Orchestrate cross-chain token transactions.
        - Supports interaction with destination chain contracts, such as distributing tokens or finalizing swaps.

The primary components it interacts with and relies upon for its functionality are:

3. **ZetaChain Gateways**:
    - **Location**: Deployed on ZetaChain and in the source/destination chains.
    - **Responsibilities**:
        - Facilitates secure token transfers and message delivery between blockchains.
        - Acts as the bridge for transferring tokens and executing cross-chain operations.
    - **Capabilities**:
        - Encapsulates and decodes cross-chain payloads.
        - Interfaces with relayers or validators to ensure proper message propagation.
4. **Relayers and Oracles**:
    - **Location**: Operates off-chain but integrated with ZetaChain Gateways contracts.
    - **Responsibilities**:
        - Facilitate cross-chain messaging by observing events on source chains and relaying them to ZetaChain or destination chains.
    - **Capabilities**:

- Enable cross-chain messaging.

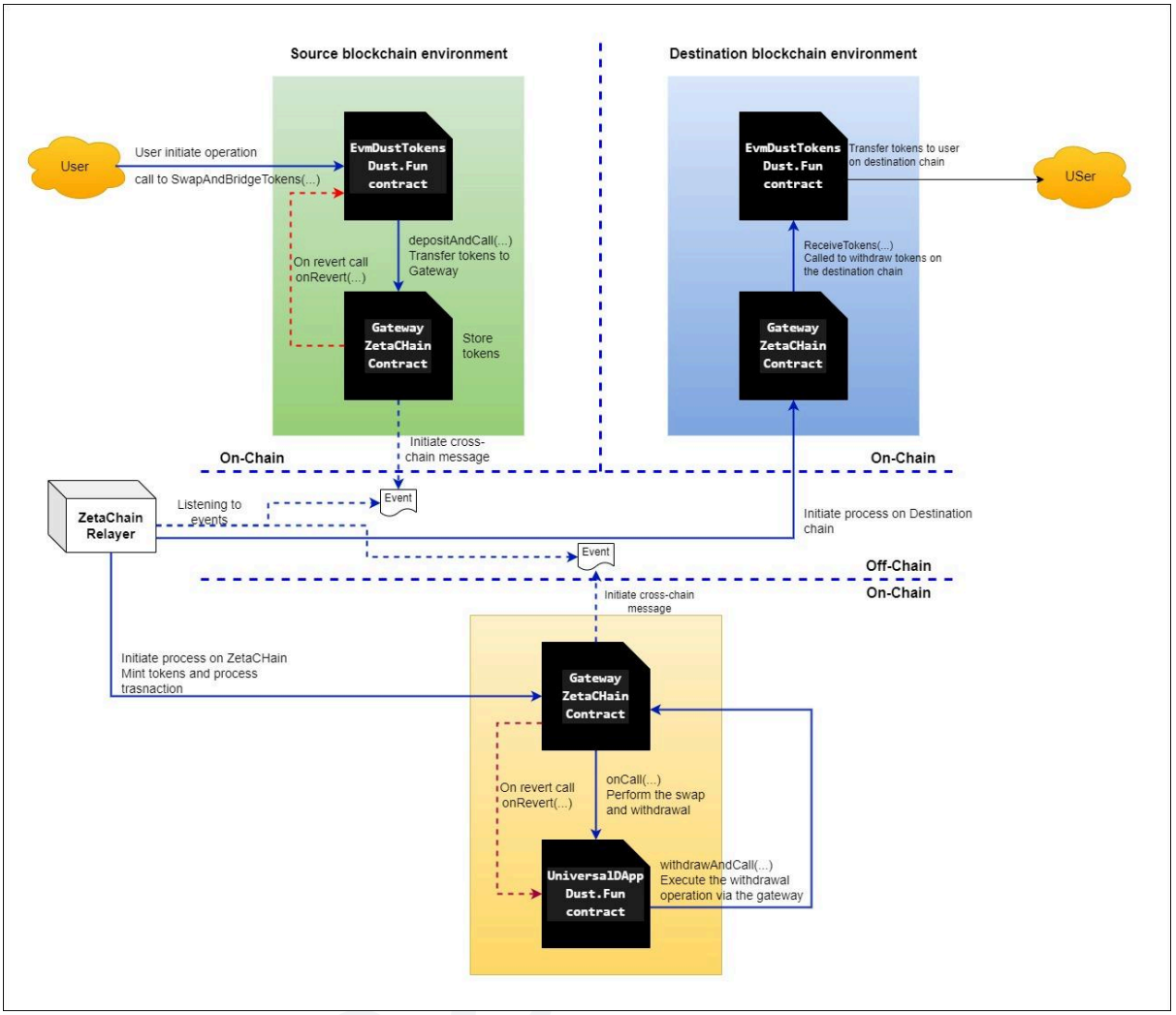

Figure I: Conceptual design of platform modules and main interactions

System Actors

1. **Users**:
   - **Responsibilities**: Initiate operations for token consolidation.
   - **Capabilities**:
     - Set the threshold for the amount that is considered dust token
     - Approve token transfers.
     - Select target tokens and destination chains for their operations.
2. **EvmDustTokens Contract Admins**:
   - **Responsibilities**:
     - Manage the whitelist of supported tokens.
     - Manage protocol fees.
     - Hold discretionary authority over refund management
3. **Relayers/Oracles**:
   - **Responsibilities**:
     - Facilitate communication between chains by observing events and transmitting messages.

Main Flow: Token Consolidation

1. **Input**: User selects tokens to consolidate and specifies a target token and destination chain.
2. **Process**:
    - EvmDustTokens contract validates user inputs and executes swaps on the source chain.
    - The contract sends the consolidated token to ZetaChain via its gateway.
    - The UniversalDApp on ZetaChain processes the tokens and sends them to the destination chain's contract.
3. **Output**: Target token delivered to the user on the destination chain via the `EvmDustTokens` contract.

Protocol dependencies

The Dust.Fun protocol is built on a modular architecture with dependencies on ZetaChain for cross-chain interoperability.

1. **Relationship with ZetaChain**:
    - The protocol heavily relies on ZetaChain for secure token transfers and cross-chain message delivery.
    - ZetaChain serves as the intermediary blockchain, enabling communication between the source and destination chains.
2. **ZetaChain Overview**:
    - **Type**: EVM-compatible blockchain designed for interoperability.
    - **Functionality**:
        - Native support for cross-chain transactions.
        - Management of ZRC-20 tokens, an extended ERC-20 standard for cross-chain operations.
    - **Technical Features**:
        - **Consensus Mechanism**: Delegated Proof of Stake (DPoS) with validator-based security.
        - **Interoperability**: Uses relayers and validators to observe and validate events across chains.

5. *Evaluation Methodology*

The evaluation methodology used in this report to rate the finding follows the principles established by the OWASP Foundation Risk Rating Methodology
This methodology is based on the definition of 3 fundamental concepts:

- Likelihood: This is a rough measure of how likely the finding vulnerability is to be uncovered and exploited by an attacker. Generally, identifying whether the likelihood is low, medium, or high is sufficient.

- Impact: The goal is to estimate the magnitude of the impact on the system if the vulnerability were to be exploited. Whenever possible, it should be considered the `Technical Impact`, loss of confidentiality, integrity, availability, accountability. And the `Business Impact`, financial and reputation damage, non-compliance, privacy violation.

- Severity: The likelihood estimate and the impact estimate are put together to calculate an overall severity for the risk. This is done by figuring out whether the likelihood and impact are low, medium or high.

Likelihood and the impact are estimated based on the following values:

	High	Medium	Low
Likelihood	The issue is trivial to exploit and the conditions that need to be met are easy to achieve.	The issue presents a moderate complexity to be executed and requires compliance with specific conditions for its application.	The issue presents a high level of complexity to be executed and requires compliance with very specific conditions for its application.
Impact	The issue can cause great damage to a large portion of users, causing monetary losses or the inability to execute critical functions, leaving the protocol in a non-functional state.	The issue can cause moderate damage affecting a fraction of users, causing monetary losses or the inability to execute some functions, leaving the protocol in an unstable state.	The issue cannot cause permanent damage to the protocol, affecting users with minor inconveniences, leaving the protocol in a recoverable state.

The following table summarizes the combination of likelihood and impact to obtain the severity of the finding:

Overall finding Severity				
Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Note *	Low	Medium
		Low	Medium	High
		Likelihood		

Some findings may not fit these categories, so the following additional categories are defined for those cases:

- **Best Practice:** This category is used to point out the possible application of smart contract development best practices that can enhance overall adherence to best practices, which can make the code more secure, readable, or auditable.
- **Informational:** This category is used to highlight areas of attention in the code that may not pose any risk but are relevant to the protocol due to their interactions or business flow;

***:** In the previous table these categories could be used in the case of a finding that has low likelihood and impact.

6. Summary of Finding

#	Finding	Classification	Status
1	Possible withdrawal of user funds in `withdrawTokenFees(...)`	Low	Acknowledged
2	Centralized management of refunds in `sendRefunds(...)` function	Low	Acknowledged
3	Unchecked address input parameter	Informational	Acknowledged
4	The `removeToken(...)` array holds residual data	Informational	Acknowledged
5	Lack of duplicate entry validation in token initialization loop	Low	Unresolved
6	Dependence Solely on ZetaChain for cross-chain operations	Medium	Unresolved
7	Unclear purpose in using the `payable` modifier in the contract constructor	Question	Unresolved

7. Finding

7.1. Possible withdrawal of user funds in `withdrawTokenFees(...)`

Files:	EvmDustTokens.sol	**Likelihood:**	Low
Severity:	Low	**Impact:**	Medium

Description: The `withdrawTokenFees(...)` function allows the contract owner to withdraw all ERC20 token balances from the contract. However, the function does not distinguish between protocol fee balances and user-deposited funds meant for refunds or other specific purposes. As a result, funds earmarked for user refunds may be inadvertently withdrawn alongside protocol fees, leaving the contract without sufficient funds to process refund requests.

Impact:

1. **Risk to user refunds:** If user-deposited funds are withdrawn as part of protocol fees, the contract will be unable to process refund requests, resulting in financial losses for users.
2. **Operational complexity:** The absence of a mechanism to segregate protocol fees from user funds complicates fund management and increases the risk of administrative errors.

Exploitation Scenarios:

1. A user mistakenly deposits tokens not whitelisted
2. The deposited amount of these tokens is added to the global refund amount
3. The contract owner uses `withdrawTokenFees(...)` to withdraw ERC20 token balances, unintentionally including user-deposited funds meant for refunds.
4. The contract cannot fulfill refund requests due to insufficient funds.

Recommendation: Consider modifying the `withdrawalTokenFees(...)` function to ensure that funds withdrawn correspond solely to platform fees, analogously to the `withdrawFees(...)` function, thus ensuring that funds deposited by users are excluded from withdrawals.
You could introduce a separate variable to explicitly track protocol fee balances for ERC20 tokens or consider deducting users' refund balances or other allocated amounts in the withdrawal process.

Status: Acknowledged

Comments from client: The protocol doesn't have any ERC20 refunds nor ERC20 deposits. All tokens held by the contract are the fee tokens. The refunds are only in a native token, and if a user accidentally sends some ERC20 tokens directly to the protocol, we won't be able to prevent that.

7.2. Centralized management of refunds in `sendRefunds(...)` function

Files:	EvmDustTokens.sol	**Likelihood:**	Low
Severity:	Low	**Impact:**	Medium

Description: The contract grants the owner centralized authority to execute the `sendRefunds(...)` function. This function allows the protocol owner to manually return tokens to users when they have been locked in the contract due to operational errors.

However, the implementation of this process is entirely manual:

- The owner determines the refund amount and recipient without any automated verification or validation mechanism.
- The system does not maintain a detailed ledger or record of blocked tokens per user. Instead, it aggregates all locked tokens as a single lump sum in the `refunds` balance.

This lack of individualized tracking introduces potential risks:

- Human errors or manipulation: Errors in manual refund allocation could occur, leading to discrepancies or unequal treatment of users. The operation depends on human interactions and operational rigor of the owner or administrative team.
- Transparency Concerns: Users have no visibility into their specific refund amounts, which could undermine trust in the protocol. Mismanagement of refunds can directly affect user funds and lead to a loss of confidence in the platform.

Recommendation: Consider implementing refund tracking per user, introducing a structure to maintain individualized refund balances for each user. This approach ensures greater transparency and allows users to monitor the status of their refund directly on-chain.

Enhance `sendRefunds(...)` logic: Consider modifying the refund operation to deduct refunds from the specific user's balance rather than a lump sum.

Increase Transparency: Allow users to query their refund status via a public function. Emit events whenever refunds are allocated or processed, to provide a clear audit trail for refunds, improving the accountability of the system.

Status: Acknowledged

Comments from client: Agree with potential risks. The `ReceiveTokens(...)` function updates refunds in 2 cases: If the output token is not whitelisted or if the recipient cannot accept native tokens. The problem with the latter is if we only allow the recipient to withdraw refunds, the refunds may be stuck forever in the contract.

7.3. Unchecked address input parameter

Files:	EvmDustTokens.sol	**Likelihood:**	Medium
Severity:	Informational	**Impact:**	Low

Description: The `constructor(...)` implements proper input validations for parameters, but does not perform validation of the `_universalDApp` parameter. It is good practice to perform input validation to mitigate human error.

```
constructor(
        IGatewayEVM _gateway,
        ISwapRouter _swapRouter,
        address _universalDApp,
        address payable _wNativeToken,
        address _initialOwner,
        IPermit2 _permit2,
        address[] memory _tokenList
    ) payable Ownable(_initialOwner) {
        if (
            address(_gateway) == address(0) ||
            address(_swapRouter) == address(0) ||
            _wNativeToken == address(0) ||
            address(_permit2) == address(0)
        ) revert InvalidAddress();
        // …
        }
    }
```

Recommendation: Consider adding the validation of `_universalDApp` parameter in the constructor.

Status: Acknowledged

Comments from client: Agree

7.4. The `removeToken(...)` array holds residual data

Files:	EvmDustTokens.sol	**Likelihood:**	High
Severity:	Informational	**Impact:**	No impact

Description: The `removeToken(...)` function allows administrators to remove addresses from the `tokenList` array, which maintains the list of token addresses accepted on the platform. The function employs low-level assembly to directly manipulate the length of the `tokenList` array in storage.

```
function removeToken(address token, uint256 index) external onlyOwner {
        // ...
        delete isWhitelisted[token];

        // ...
        uint256 len = tokenList.length - 1;
        tokenList[index] = tokenList[len];
        assembly {
            sstore(tokenList.slot, len)
        }
        // ...
    }
```

While this operation adjusts the logical size of the array, it does not explicitly remove the last element. Consequently, residual or orphaned values remain in the array's storage, and these values can still be accessed if assembly is used or if the length of the array is changed again.

Impact:

The `tokenList` array is a critical structure used by various system functions to access the list of accepted tokens. Improper handling of this array can lead to:

- Residual Data Access: Functions manually accessing array values could find spurious values.
- Logic Errors: Future system updates or external interactions using assembly could unintentionally misuse the array, leading to subtle and hard-to-diagnose errors.

Likelihood:

The likelihood of this issue affecting current functionality is low and mitigated due to the system's use of the `isWhitelisted` mapping to track active tokens. This mapping serves as the primary validation mechanism, ensuring that only whitelisted tokens are considered valid. However, the potential for errors increases if future changes rely on the array itself rather than the mapping.

Recommendation: To address this and maintain code robustness consider using Solidity's native `pop(...)` method to remove the last element of arrays in the `removeToken(...)` function. This ensures that the logical and physical state of the array are synchronized, eliminating residual values.

Status: Acknowledged

Comments from client: We agree in general, but in our case there is no impact. Our contract is not a proxy and doesn't access the storage directly anywhere else. Additionally, if we add a new whitelisted token it will overwrite the old value and will be cheaper, because we will overwrite a non-zero storage slot.

7.5. Lack of duplicate entry validation in token initialization loop

Files:	EvmDustTokens.sol	**Likelihood:**	Low
Severity:	Low	**Impact:**	Medium

Description: In the `constructor(...)`, tokens provided via the `_tokenList` parameter are added to the `tokenList` array and marked as whitelisted using the `isWhitelisted` mapping. However, the current implementation does not validate for duplicate entries in `_tokenList`, consequently, the same token address can be added multiple times to the `tokenList` array, resulting in redundant storage entries and potential inconsistencies in future operations.

```
constructor(
    /// ...
    address[] memory _tokenList
)
payable Ownable(_initialOwner)
{
    /// ...
    uint256 tokenCount = _tokenList.length;
    address token;
    for (uint256 i; i < tokenCount; ++i) {
        token = _tokenList[i];
        if (token == address(0)) revert InvalidToken(token);
        isWhitelisted[token] = true;
        tokenList.push(token);     /// Potentially adds duplicate tokens
        /// ...
        }
    }
```

In contrast, the `addTokens(...)` function, which performs a similar operation, includes a duplicate check to ensure that tokens already whitelisted cannot be added again.

```
function addTokens(address[] calldata tokens) external onlyOwner {
        uint256 tokenCount = tokens.length;
        address token;
        for (uint256 i; i < tokenCount; ++i) {
            token = tokens[i];
            if (token == address(0)) revert InvalidAddress();
            if (isWhitelisted[token]) revert TokenIsWhitelisted(token);
            isWhitelisted[token] = true;
            tokenList.push(token);
```

```
            emit TokenAdded(token);
        }
    }
```

The absence of such a check in the constructor introduces design inconsistencies and may lead to unexpected behavior.

Impact

Storage Inefficiency: Duplicate token entries in the `tokenList` array increase storage costs unnecessarily, particularly during deployment. This inefficiency can also extend to any functions that iterate over the array.

Operational Complexity: When the owner uses the `removeToken(...)` function to remove a token, duplicate entries in the `tokenList` may prevent the complete removal of the token, potentially causing significant disruptions in the protocol's functionality. Resolving this issue would require multiple removal operations for the same token, increasing transaction costs and operational overhead..

Likelihood

The likelihood of this issue is mitigated by the fact that this is an administrative function and only occurs during the construction process.

Exploitation Scenarios

1. **Deployment Phase**:
    - If the `_tokenList` parameter includes duplicate token addresses, the same token is added multiple times to the `tokenList` array during the contract's deployment.
2. **Token Removal**:
    - When the owner removes a token using `removeToken(...)`, duplicate entries may remain, leading to unexpected results or inconsistencies in subsequent operations

Recommendation: Consider encapsulating the logic for adding tokens into a dedicated helper function, to be utilized by both the constructor and the `addTokens(...)` function. The helper function should include validation to prevent duplicate entries. Depending on the desired behavior, it should either revert the transaction upon encountering a duplicate token or skip the duplicate and continue processing the remaining tokens.

Centralizing this logic eliminates the redundancy of maintaining identical code in multiple locations and enhances the maintainability of the codebase.

Status: Unresolved

Comments from client:

7.6. Dependence Solely on ZetaChain for cross-chain operations

Files:	EvmDustTokens.sol UniversalDApp.sol	**Likelihood:**	Low
Severity:	Medium	**Impact:**	High

Description: The **Dust.Fun** protocol heavily relies on ZetaChain for its cross-chain functionality, including token bridging and communication between source and destination chains. ZetaChain acts as an intermediary layer facilitating the transfer of tokens and execution of cross-chain operations through its relayers, gateways, and messaging infrastructure. This dependency introduces a significant risk to the Dust.Fun protocol if ZetaChain were to experience operational disruptions or permanently cease to function.

Potential Risks:

1. **Operational Disruption**:
   If ZetaChain temporarily halts its operations due to technical or governance issues, the Dust.Fun protocol's cross-chain features would become unavailable, affecting users' ability to bridge or consolidate their tokens.
2. **Permanent Shutdown**:
   In the event that ZetaChain ceases to operate entirely, the Dust.Fun protocol would lose the functionality to manage cross-chain transactions, effectively rendering these features inoperable.
3. **User Funds at Risk**:

   Tokens already transferred to ZetaChain (as part of ZRC-20 wrapping) would become inaccessible if the ZetaChain infrastructure is no longer operational.

   Users holding funds mid-transaction (e.g., during bridging or swapping) may face challenges recovering their assets without manual intervention.

Impact

1. **Loss of Cross-Chain Functionality**:
   The protocol's primary purpose, aggregating and converting fragmented balances across multiple chains, would no longer function without ZetaChain. This dependency significantly limits the protocol's resilience in case of ZetaChain failure.
2. **User Trust**:
   A failure to recover user funds in the event of ZetaChain disruption would erode trust in the protocol, potentially leading to reputational damage and reduced user adoption.
3. **Inaccessible Funds**:
   Tokens locked on ZetaChain (e.g., as ZRC-20 assets) would remain unrecoverable, resulting in financial losses for users unless manual intervention.

Recommendation: To mitigate the risks associated with dependence on ZetaChain, consider incorporating measures similar to the following:

1. **Decentralized Backup Mechanisms**:
   Consider designing an off-chain mechanism or decentralized framework that tracks user balances and transaction states, allowing users to verify and claim their tokens on the source or destination chains in the event of ZetaChain failure.
2. **Fallback Recovery Contracts**:
   Deploy contracts on source and destination chains to store additional metadata about token transfers. This could include:
   - Proofs of token deposits or withdrawals.
   - Transaction states for cross-chain operations.

   These contracts would serve as a basis for resolving incomplete transactions and manually releasing locked funds if ZetaChain becomes unavailable.

3. **Transparent User Balance Records**:
   Maintain an on-chain ledger (accessible to users) that tracks deposited and bridged funds. This ledger could enable users to reclaim their funds from source contracts if bridging fails due to ZetaChain disruption.
4. **Contingency Planning**:

   Evaluate the possibility of integrating a secondary cross-chain solution as a backup to ensure continuity.

5. **Enhanced User Education**:
   Provide clear guidance to users on how funds are managed during cross-chain operations, the risks involved, and steps to recover assets in case of disruptions.

Status: Unresolved

Comments from client:

7.7. Unclear purpose in using the `payable` modifier in the contract constructor

Files:	UniversalDApp.sol	**Likelihood:**	Low
Severity:	Informational	**Impact:**	Low

Description: The `UniversalDApp` constructor includes the ``payable keyword, allowing it to receive Ether during deployment. However, there doesn't appear to be any functionality within the constructor or the contract that utilizes the native token. Could you clarify the reasoning behind its inclusion?

Some potential concerns with this approach include:

- Funds sent during deployment would remain as part of the contract's balance without any defined mechanism for withdrawal or utilization.
- The inclusion of payable might introduce unnecessary ambiguity into the contract design

Recommendation: Is this modifier intentionally included for future extensibility, or is it an oversight? If it is not required, would it be advisable to remove it to enhance clarity and reduce potential risks?

Status: Unresolved

Comments from client:

8. Documentation Evaluation

Proper documentation is a cornerstone of any robust protocol, serving as a foundation for its security, usability, and long-term evolution. Comprehensive documentation encompasses multiple forms, each tailored to specific audiences and purposes:

Core Documentation:

- **White Paper**: Provides an overarching view of the protocol's purpose, design, and intended use cases. This is crucial for stakeholders, investors, and the broader community to understand the protocol's vision and objectives.
- **Yellow Paper**: Delivers a technical and mathematical specification of the protocol, outlining its algorithms, mechanisms, and formal definitions. This ensures clarity and rigor in the protocol's design, enabling technical validation and analysis.

User-Focused Documentation:

- **User Manuals**: Simplify onboarding by offering clear, step-by-step guidance for users. This is critical for lowering barriers to entry, fostering adoption, and reducing user errors that could compromise security or functionality.

Developer-Centric Documentation:

- **Commented Code**: Embedding meaningful comments within the codebase enhances its readability, facilitates understanding, and reduces the likelihood of bugs or misinterpretations by developers.
- **API References and Developer Guides**: Ensure that external contributors and integrators can seamlessly interact with the protocol, driving ecosystem growth and enabling innovation.

Relevance to Security:

- Clear and detailed documentation aids in identifying and addressing potential vulnerabilities. Auditors rely heavily on well-documented specifications to understand the protocol's intended behavior and ensure alignment with its implementation.
- Poor or incomplete documentation increases the risk of overlooked issues, misconfigurations, or unintended behaviors during audits or future upgrades.
- Comprehensive documentation accelerates the learning curve for new users, developers, and contributors, fostering a more inclusive ecosystem.
- It ensures that knowledge is not siloed within a few individuals, reducing dependency on the original team and enhancing the protocol's resilience.

Thorough documentation is not merely a supplementary aspect of protocol development but an important component of its success. It enhances security, ensures usability, and provides a foundation for the protocol's sustainability, scalability, and future innovation. A well-documented protocol is a sign of maturity, transparency, and a commitment to building a reliable ecosystem.

The **Dust.Fun** team provided documentation explaining the behavior of the core protocol, its components, and its supported cross-chain functionality. This documentation was instrumental in achieving understanding of the protocol's intended behavior and overall architecture.

Protocol Description:

The documentation provides a high-level overview of the protocol's purpose, such as consolidating "dust" tokens and enabling cross-chain operations.

Technical Details:

Some technical aspects, such as interactions with ZetaChain, contract functions, and supported chains, are addressed. However, the documentation lacks granular details like in-depth explanations of:

- Contract logic and its interaction with ZetaChain and other layers.
- Formal specifications of the protocol's mechanisms, comparable to Yellow Paper-level documentation.
- Comprehensive risk assessments for dependencies, particularly for critical systems like ZetaChain.
- Formalized descriptions of expected behaviors and identified edge cases to assist in testing and validation.

Developer Documentation:

There is no dedicated API reference, deployment guides, or detailed instructions for developers wishing to interact with the protocol.

Recommendations for Improvement

To meet industry standards and support a broader audience of users, developers, and auditors, the documentation could be improved by:

- Including comprehensive risk assessments for dependencies like ZetaChain, highlighting potential vulnerabilities and mitigation strategies.
- Providing formalized descriptions of expected system behaviors, edge cases, and failure modes to enhance system reliability.
- Developing developer-centric documentation, such as detailed APIs, integration examples, and a clear deployment pipeline.

9. Summary

The **Dust.Fun** protocol has a solid foundation for its intended purpose of simplifying wallet management by consolidating "dust" tokens. However, addressing the identified issues, particularly those related to documentation, centralized components, and dependencies, is essential to enhance its security posture, operational transparency, and overall robustness.

The recommendations provided in this report aim to guide the Dust.Fun team in improving the protocol's security and usability. Further reviews and ongoing testing are encouraged as the protocol evolves, especially given its reliance on emerging cross-chain technologies like ZetaChain.

10. About Rather Labs

Founded in July 2020 in Argentina, Rather Labs has established itself as a premier blockchain technology hub, driven by the entrepreneurial vision of its founders and a relentless pursuit of innovation. With a team of over 130 professionals, we have partnered with more than 40 entities across 13+ countries, delivering advanced blockchain platforms, financial applications, and tailored solutions that empower our clients to succeed in the rapidly evolving Blockchain Ecosystem.

At the heart of our offerings is Blockchain Solutions Provider, a comprehensive development outsourcing service that takes projects from concept to mainnet. We specialize in crafting scalable, secure, and sustainable platforms that align with our clients' goals. Complementing this is our Research and Development department, which pioneers advancements in blockchain ecosystems, building proof-of-concepts, developer tools, and innovative technologies. Additionally, our Talent Headhunting service ensures access to top-tier professionals, tailored to the unique needs of each project.

Our expertise extends to Blockchain Audits, where we ensure the security and reliability of smart contracts and decentralized applications across 20+ blockchains. A prime example is our collaboration with Armada Music, We were tasked with auditing their smart-contract for a drop of Non-Fungible Tokens (NFTs). To ensure the code for Armada Music's NFT drop was mainnet-ready, our team conducted a thorough manual inspection. Throughout the process, we kept their team informed with daily reports and maintained open lines of communication through multiple channels.

Our dedication to excellence is reflected in notable partnerships. With Membrane Labs, we enhanced OTC trading and lending platforms, facilitating $3 billion in loans for financial institutions. Similarly, our collaboration with Hatom Protocol led to the creation of MultiversX's first decentralized lending and borrowing platform, achieving an impressive $250M in Total Value Locked (TVL). These successes underscore our expertise in delivering impactful, scalable solutions while serving as a trusted partner in blockchain innovation.

You can find more info and some of our more relevant projects on our Company Deck

11. Disclaimer

This report has been prepared based on the materials and documentation provided to us for the purpose of conducting the security review outlined in sections 1. Report Summary and 3. Audited Files (specified in the audit services agreement signed between the Parties). The findings in this report may not encompass all vulnerabilities or security concerns, as the analysis was conducted within the specified scope and under the conditions provided, and RL is not liable for the vulnerabilities not found in this audit report.

The review and this report are provided "as is," "where is," and "as available," without any guarantees or warranties. You acknowledge that the access to or use of this report, including any associated services, products, protocols, platforms, content, or materials, is entirely at your own risk. Blockchain technology is still evolving and subject to unforeseen risks and vulnerabilities. This review does not extend to areas such as the compiler layer or any elements beyond the programming language that could present security risks.

This report does not constitute an endorsement of any particular project or team, nor does it guarantee the security of the reviewed codebase or its associated components. It is intended solely for informational purposes and should not be relied upon by any third party, including for making decisions related to the purchase or sale of any products, services, or assets.

To the maximum extent permitted by law, we disclaim all liability related to this report, its content, and any associated services or products. This includes, but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We make no representations, endorsements, guarantees, or assurances regarding any product or service mentioned in the report, any open-source or third-party software, code, libraries, materials, or information linked to or referenced by the report, or any related services and products. Furthermore, we assume no responsibility for any interactions, transactions, or disputes between you and third-party providers of products or services.

As with any decision involving the purchase or use of a product or service, you are encouraged to exercise caution and apply sound judgment.

FOR THE AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, AND ANY RELATED USAGE OR SERVICES SHALL NOT BE INTERPRETED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.