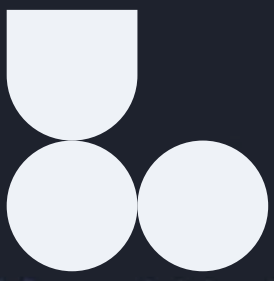
07/2023

Audit report for



yet another defi

scalablesolutions.io

scalable

Scope

The scope of this audit includes a comprehensive review of the following smart contracts located in the respective files:

CONTRACTS/SRC/SAIL/EXTERNAL

SailFlashLallet.sol

Provides functionality to perform instant exchanges, allowing users to borrow assets from liquidity providers without the need to provide initial collateral.

SailRFQTransformer.sol

Facilitates request for quote (RFQ) orders by attempting to fill orders through external calls to specific contracts and by providing fallback mechanisms in case of failures.

CONTRACTS/SRC/SAIL/MIGRATIONS

SailArbitrumMigration.sol

Enables the migration of assets from the main Ethereum network to the Arbitrum network, providing a seamless transition for users wishing to take advantage of Arbitrum's scaling Layer-2 solution.

SailMigration.sol

Manages the deployment and registration of features in the 0x contract, ensuring that the 0x protocol is properly initialized and configured with the desired set of features.

CONTRACTS/SRC/SAIL/FEATURES

AnyRecipientFeature.sol

Provides the ability to send tokens to any recipient address, which provides greater exibility when transferring tokens and interacting with other smart contracts.

SailAdapterFeature.sol

Serves as an adapter for interfacing with various decentralized exchanges, enabling seamless integration and interoperability between the 0x protocol and protocols utilized on exchanges.

SailUniswapV3Feature.sol

Provides interoperability with the Uniswap V3 protocol, offering advanced liquidity management and trading options for users.

The scope of the audit includes checking the codebase of contracts, evaluating their security measures, evaluating compliance with best practices, and identifying potential vulnerabilities or risks associated with their implementation. The goal is to ensure the overall reliability and functionality of these contracts, complementing the capabilities of the 0x protocol and improving the user experience when interacting with decentralized exchanges.

Methodology

The methodology employed is not a rigid formal process. Instead, it is a blend of diverse methods and tactics, tailored and adapted uniquely for each individual project. Factors such as the project's structure, the technologies used, and the client's expectations from the audit all contribute to shaping the methodology. In the present audit, we utilize:

Overall Code Score

The code undergoes a review for clarity, consistency, style, and compliance with the coding practices pertinent to the specific programming language in use. This includes examining indentation consistency, naming conventions, commented code blocks, instances of code duplication, and the presence of confusing names or ambiguous, irrelevant, or absent comments. At this phase, a general comprehension of the code's structure is achieved.

Entity Utilization Analysis

This analysis involves examining the use of various entities defined within the code. This not only includes their internal usage within different parts of the code but also potential external applications. We ensure that the objects are defined appropriately and that their scopes and access levels are current. At this point, we gain an understanding of the overall system architecture and the relationships between various parts of the code.

External Access Control Evaluation

In this stage, we focus on those entities that are externally accessible. We thoroughly analyze the implemented access control measures, ensuring they are current and correctly enforced. This understanding offers insights into user roles, permissions, and the system assets that require protection.

Functional Logic and Efficiency Analysis

During this phase, we scrutinize the logic of individual functions for their correctness and efficiency. We confirm that the code functions as intended, the algorithms are both optimal and correct, and that appropriate data types are being utilized. An examination of external libraries incorporated within the code is also conducted to ensure they are current and aptly suited to the tasks they perform. This stage grants us an understanding of the employed data structures and the purposes they serve.

Risk Level Classification

INFORMATIONAL	Informational vulnerabilities do not pose an immediate risk to a smart contract or the interacting system. Typically related to issues in documentation or code readability, these vulnerabilities may not present a direct risk, but can negatively impact code maintenance and future development.
LOW	Low-level vulnerabilities, while minor, could cause inefficiencies or unexpected contract behavior. These are not considered critical but might include, for example, misuse of a library function or a minor logic error.
MEDIUM	More severe than low-level vulnerabilities, medium-level vulnerabilities have the potential to cause minor breaches in contracts. An example could be inadequate input validation, which might allow attackers to manipulate contract behavior.
HIGH	High-level vulnerabilities represent serious threats that could lead to significant financial losses or reputational damage. These vulnerabilities could involve insufficiently protected contract features that allow unauthorized actions or fund theft by an attacker
CRITICAL	As the most severe category, critical vulnerabilities could have disastrous consequences. A contract susceptible to a replay attack, allowing an attacker to withdraw all contract funds, is one example. Another is a logic error in the contract that could be exploited to bypass access controls, enabling an attacker to seize complete control of the contract.

Findings

TOTAL ISSUES: 10

Based on the potential impact and level of risk, we have categorized the identified issues into severity levels:

INFORMATIONAL — 6 ISSUES

SailAdapterFeature.sol
Limited error handling.

SailAdapterFeature.sol
There is no description of the function.

SailArbitrumMigration.sol
No SPD license ID.

SailMigration.sol
No SPD license ID.

SailRFQTransformer.sol
No SPD license ID.

SailFlashWallet.sol
There is no description of the function.

LOW — 3 ISSUES

SailFlashWallet.sol ✓ Fixed
Unexpected result when using an argument.

AnyRecipientFeature.sol
No validation of input parameters.

SailAdapterFeature.sol ✓ Fixed
No function parameter validation.

HIGH — 1 ISSUE

SailUniswapV3Feature.sol
No validation of input parameters.

It is strongly advised that high-severity issues are addressed immediately due to the significant risk they pose to both the security and functionality of the project. Medium-severity issues should also be promptly resolved to diminish potential vulnerabilities. Even though minor issues are less critical, addressing them will enhance code readability and adherence to best practices.

Upon receiving our audit report, the project team took swift action to rectify several low-level vulnerabilities. They tackled this task with complete responsibility and seriousness.

However, a decision was made to leave the high-level vulnerability unaddressed, thereby accepting all associated risks.

Detailed Results

SailUniswapV3Feature.sol

No checks on input parameters (HIGH SEVERITY):

The *_sailUniswapV3Swap* private function called by external functions of the *SailUniswapV3Feature* contract does not check pool addresses passed to the pools argument.

_sailUniswapV3Swap

```
669    uint256 []    memory pools
```

This results in a critical vulnerability where an attacker, by imitating the swap function in their contract, can implement their own logic when funds are withdrawn from the proxy contract's address. Although, in principle, the proxy contract should not hold funds, any tokens sent to it accidentally can be withdrawn using this method.

Recommendation:

To mitigate this security risk, we recommend checking inside the V3Swap_sailUniswap function if the address being passed is actually a Uniswap pool. This would help to ensure that the transaction is carried out as expected.

Team Response:

While this vulnerability does indeed present a potential risk, it is important to note that it only allows funds to be withdrawn from the proxy contract, where funds are not stored by design. If funds are randomly sent, a rescueFunds function is available that enables the withdrawal of funds from the proxy. This means that the sender's funds aren't in danger, as the project's logic does not facilitate storing client funds on the proxy contract. Even if funds ended up there, they could be returned.

Adding additional checks would significantly increase the transaction cost, which is counterproductive as the main purpose of this functionality is to maintain low transaction costs.

SailFlashWallet.sol

Unpredictable result when using an argument (LOW SEVERITY) (FIXED):

The isETH argument in the *executeAggRouterCall* function is to indicate whether the tokenIn is an ETH address or not. However, using this argument can produce unpredictable results if the tokenIn address passed does not match the value specified in isETH.

executeAggRouterCall

```
43    bool isETH
```

Recommendation:

It is recommended to revise the use of the isETH argument. Instead, it would be more reliable and predictable to check the token type by comparing the tokenIn address with the ETH address directly inside the function. The ETH address to compare with should be: 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeeEEeE.

AnyRecipientFeature.sol

Lack of input parameter checks (LOW SEVERITY):

The *anyRecipientTransformERC20* function does not validate the input of the provided parameters. Input validation is an important security practice to ensure that input meets expected criteria and to prevent unexpected behavior or misuse. Without proper input validation, a function can accept invalid or malicious input, leading to potential vulnerabilities.

anyRecipientTransformERC20

```
21    address payable recipient,
22    IERC20TokenV06 inputToken,
23    IERC20TokenV06 outputToken,
24    uint256 inputTokenAmount,
25    uint256 minOutputTokenAmount,
26    ITransformERC20Feature.
      Transformation [] memory
      transformations
```

Recommendation:

It is recommended to incorporate input validation code that verifies the given input before proceeding with any further logic. This should encompass validation for the recipient address, input token address, output token address, the quantity of the input token, minimum output token amount, and the presence of transformations. By validating input data, the contract can establish necessary restrictions and safeguard against potential security threats linked with invalid or malevolent data.

Team Response:

Given that any RecipientTransformERC20 represents an enhanced version of TransformERC20Feature::transformERC20 (_transformERC20Private), all original side-effect verifications from the initial 0x code are also conducted. This implies that we put our trust in the original 0x design and their audits, and refrain from altering their source code. However, we acknowledge that this address could correspond to a smart contract (SC) rather than an externally owned account (EOA) due to potential partner integrations.

SailAdapterFeature.sol

No function parameter check (LOW SEVERITY) (FIXED):

The *aggRouterSwapWithFee* function is missing a check for the feePercentage parameter, which represents the commission percentage in basis points (i.e., 0.01%). Without this check, there is a potential risk of commission miscalculation due to incorrect or unexpected values.

aggRouterSwapWithFee

```
117     uint16 feePercentage
```

Recommendation:

Consider implementing a check at the beginning of the aggRouterSwapWithFee function to ensure that feePercentage is within the expected range of values.

Limited error handling (INFORMATIONAL SEVERITY):

The *_transferERC20TokensFrom* function checks the success of the transferFrom call and cancels if the call fails. However, it lacks detailed error messages. This limited error-handling approach can hinder user experience and make it difficult to identify and resolve transmission failures or exception scenarios.

Recommendation:

The error-handling mechanism in the function should be enhanced for better user feedback. Consider implementing specific error codes or messages to inform the caller about various error conditions. This would not only improve the user experience but also assist in pinpointing issues during debugging. Furthermore, logging pertinent error information could simplify the troubleshooting process significantly.

No function description (INFORMATIONAL SEVERITY):

The *_executeOutputTokenTransfer* function does not have a description block or comments explaining its purpose, expected behavior, and any associated limitations. The lack of documentation makes it difficult for auditors and other developers to understand the intended functionality. Without clear documentation, it is difficult to assess the correctness and safety of the code, which can lead to misunderstandings or missing critical issues.

Recommendation:

It is recommended to provide a clear and concise description of the feature. The description should explain the purpose of the feature, its expected behavior, supported token types (ERC-20 or ETH), and any associated restrictions. Including this information will improve code readability and help other developers, auditors, and maintainers understand the role of the feature in the contract.

SailArbitrumMigration.sol

No SPDX License Identifier (INFORMATIONAL SEVERITY):	The contract does not include an SPDX License Identifier, which is recommended for clarity of licensing terms and compliance with open-source licensing requirements.
Recommendation:	It is recommended to include an SPDX License Identifier in the contract to unequivocally set the licensing terms that dictate the distribution of their code. The addition of such an identifier enhances transparency, aiding others in understanding the permissions and restrictions associated with the contract. Furthermore, it ensures compliance with open-source licensing requirements.

SailMigration.sol

No SPDX License Identifier (INFORMATIONAL SEVERITY):	The contract does not include an SPDX License Identifier, which is recommended for clarity of licensing terms and compliance with open-source licensing requirements.
Recommendation:	It is recommended to include an SPDX License Identifier in the contract to unequivocally set the licensing terms that dictate the distribution of their code. The addition of such an identifier enhances transparency, aiding others in understanding the permissions and restrictions associated with the contract. Furthermore, it ensures compliance with open-source licensing requirements

SailRFQTransformer.sol

No SPDX License Identifier (INFORMATIONAL SEVERITY):	The contract does not include an SPDX License Identifier, which is recommended for clarity of licensing terms and compliance with open-source licensing requirements.
Recommendation:	It is recommended to include an SPDX License Identifier in the contract to unequivocally set the licensing terms that dictate the distribution of their code. The addition of such an identifier enhances transparency, aiding others in understanding the permissions and restrictions associated with the contract. Furthermore, it ensures compliance with opensource licensing requirements.

No function description (INFORMATIONAL SEVERITY):	The function named *_executeFallback* does not contain a description or explanatory comments, making it difficult to understand its intended functionality and purpose without further analysis.
Recommendation:	To improve the readability and maintainability of the code, it is recommended to provide a description or explanatory comments for the function. The description should articulate the function's purpose, its anticipated behavior, and any essential considerations or limitations. This documentation will aid developers in comprehending the function's role within the contract and will simplify subsequent maintenance and debugging processes.

Conclusion

The audit of the project's smart contracts highlighted several areas of concern that must be addressed to ensure the security, dependability, and optimal performance of these contracts. Several issues of low and informational severity were identified, in addition to one high severity issue. However, it is critical to point out that the design logic doesn't afford attackers the ability to exploit the detected high-severity vulnerability.

The project team promptly reacted, rectifying some of the identified problems. As for the other issues, no modifications are needed. It's worth mentioning that informational-level vulnerabilities do not necessitate contract re-creation.

The project team demonstrated a commendable level of diligence and responsibility in addressing the identified issues. Nonetheless, it is advised to maintain continuous monitoring and regular security checks to ensure the ongoing security and reliability of the contracts.