Audit Report

Router DexSpan

DRAFT – DO NOT PUBLISH

v0.3

May 10, 2024

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Kailaasa Infotech Pte Ltd to perform a security audit of the Router DexSpan smart contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/router-protocol/asset-forwarder-contracts
Commit	`93dcff0884c3a03d2e18dfc0398d5d4e1f26046f`
Scope	Only contracts in `evm/src/dexspan` were in scope.
Fixes verified at commit	`144f352288fe85c40b3de198d671904fcdf447d5` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

DexSpan is a swap-aggregator with the versatile functionality of facilitating token exchanges before and after token bridging operations. Furthermore, it is equipped with message-passing capabilities, channeled through other Router components like the Asset Bridge and the Asset Forwarder.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium	The provided documentation does not cover all functionalities implemented in the codebase.
Test coverage	Low	There are no tests in place for DexSpan.

Summary of Findings

No	Description	Severity	Status
1	Missing transaction revert could lead to loss of funds	Critical	Acknowledged
2	Contract upgradability is not properly implemented	Major	Resolved
3	The UniswapV2's skim address is hardcoded which may lead to a loss of funds on other chains	Major	Resolved
4	Maximum approval on third-party components may lead to drain of funds	Major	Resolved
5	DEX swap functions enable users to perform arbitrary calls	Major	Acknowledged
6	Deposits in native currency always revert	Major	Resolved
7	Balance retrieval logic enables attackers to drain contract funds	Minor	Acknowledged
8	Missing emission of events for configuration changes	Minor	Resolved
9	Missing addresses validation	Minor	Resolved
10	Lack of contract ownership transfer functionality	Minor	Partially Resolved
11	Anomalies in hexadecimal flags' interval width can lead the transaction to revert	Minor	Resolved
12	The swap function accepts ETH even if not defined in the token swap array leading to funds stuck in the contract	Minor	Resolved
13	The `assetBridge` and `assetForwarder` could drain funds from the contract by sending a faulty swap message	Minor	Acknowledged
14	Insecure non-production contracts in the codebase	Minor	Resolved
15	Missing role segregation between `DEFAULT_ADMIN_ROLE` and `FACTORY_SETTER_ROLE`	Minor	Resolved
16	Avoid using `extcodesize` to check for externally owned accounts	Minor	Acknowledged

17	Lack of input validation on `_swapMultiInternal` function parameters	Informational	Resolved
18	Missing UniswapV2 pair existence check	Informational	Resolved
19	Functions called externally are defined as `public`	Informational	Resolved
20	Use custom errors for gas efficiency	Informational	Acknowledged
21	Unused functions, variables, parameters, and return values	Informational	Acknowledged
22	Altered `_returnData` in Multicall contract	Informational	Resolved
23	Initializing variables to their default value is inefficient	Informational	Resolved
24	Use `++i` instead of `i++` to save gas	Informational	Acknowledged
25	Contracts should implement a two-step role ownership transfer	Informational	Resolved
26	Code duplication for `handleMessage` and `handleAssetBridgeMessage`	Informational	Resolved
27	Duplicated computation leads to inefficiency	Informational	Resolved
28	`swapInSameChain` and `swapMultiWithRecipient` functions should emit different events	Informational	Resolved
29	Use of magic numbers decreases maintainability	Informational	Acknowledged
30	Redundant parameter `minReturn` in `_swap` function	Informational	Partially Resolved
31	Missing usage of the `IBridge` interface	Informational	Acknowledged

Detailed Findings

1. Missing transaction revert could lead to loss of funds

Severity: Critical

In `evm/src/dexspan/DexSpan.sol:421-437`, the `messageHandler` function should transfer funds to the `recipient` and, if `isInstruction` is set to `true`, it initiates a `call` to the `recipient`.

However, it does not check the `execFlag` and does not roll back the current state if the call fails.

As a consequence, this could potentially result in the loss of funds within the `recipient` contract.

Recommendation

We recommend implementing a mechanism to ensure the state is rolled back in the event of a failed call to the recipient when `isInstruction` is `true`.

Status: Acknowledged

The client states that the issue only affects the interaction with third-party contracts.

2. Contract upgradability is not properly implemented

Severity: Major

The `DexSpan` contract is `Initializable` and intended for upgrades, using an initializer function instead of a constructor.

However, it implements a `constructor`.

As highlighted in the Open Zeppelin documentation on Writing Upgradeable Contracts, this misuse of functions would make the contract not upgradeable.

Recommendation

We recommend implementing an `initializer` function, removing the `constructor`, and using a `UUPS` proxy upgrade pattern.

Status: Resolved

The client removed the support to `Initializable` since they are not planning to make the contract upgradable.

3. The UniswapV2's skim address is hardcoded which may lead to a loss of funds on other chains

Severity: Major

The UniswapV2's skim contract address is hardcoded in the constant `skimAddress` in `evm/src/dexspan/DexSpan.sol:76` and `evm/src/dexspan/DexSpanView.sol:45`.

However, since the skim contract address may vary on different chains, the hardcoded address may result in a loss of funds on another chain where the address does not belong to Uniswap.

Recommendation

We recommend avoiding hardcoded addresses and instead assigning them during deployment, initialization, or using a specific setter function.

Status: Resolved

4. Maximum approval on third-party components may lead to drain of funds

Severity: Major

The `_swapOnOneInch`, `_swapOnParaswap`, and `_swapOnUniswapV3` functions defined respectively in `evm/src/dexspan/DexSpan.sol:593-615`, `evm/src/dexspan/DexSpan.sol:617-643`, and `evm/src/dexspan/DexSpan.sol:564-591` allow swapping tokens on different DEXs.

However, they do not limit token approvals appropriately, by using a `type(uint256).max` approval to the addresses of those protocols each time they are invoked.

As a result, this could lead to a drain of funds if there is a bug or malicious implementation in the allowed contracts.

Recommendation

We recommend avoiding giving maximum allowance to third-party components and instead approving the required amounts for each transaction.

Status: Resolved

5. DEX swap functions enable users to perform arbitrary calls

Severity: Major

The functions `_swapOnOneInch`, `_swapOnParaswap` and `_swapOnUniswapV3` defined in `evm/src/dexspan/DexSpan.sol:593-615`, `evm/src/dexspan/DexSpan.sol:617-643`, and `evm/src/dexspan/DexSpan.sol:564-591`, respectively, allow swapping tokens on different DEXs.

However, since these functions rely on unverified user input `_data` to call any given function of the address mapped at the input-provided `flag`, users can call arbitrary functions to let the transaction succeed even if no swap is performed.

Moreover, in the case of a multihop swap, parameter calculations occur dynamically during the process, making it challenging for users to define the correct parameters in the `_data` field. This might lead to a loss of funds and leftover funds stuck in the contract.

We classify this issue as major instead of critical since the frontend is assumed to correctly and safely construct messages.

Recommendation

We recommend restricting the function call to a specified selector, reducing arbitrary input data, and in case of multihop swaps, passing the parameters calculated in the process to the defined swap functions.

Status: Acknowledged

6. Deposits in native currency always revert

Severity: Major

The `swapAndDeposit` function allows users to deposit native currency or swap tokens to native currency and deposit them to the asset forwarder contract.

Then it calls the `_swapMultiInternal` function. If there is a swap whose final token is the native currency, the contract will withdraw the funds from the native wrapper contract.

However, it does not transfer funds when calling the deposit function, as this function only approves normal ERC20 contracts.

Consequently, since the asset forwarder checks the `msg.value`, such transaction always revert.

Recommendation

We recommend implementing a different way to deposit native currency, allowing the contract to send the appropriate value, or implementing a consistent pattern to avoid depositing native currency.

Status: Resolved

7. Balance retrieval logic enables attackers to drain contract funds

Severity: Minor

In `evm/src/dexspan/DexSpan.sol:352`, `452`, and `465`, the `universalBalanceOf(address(this))` function should fetch the balances used for computing transfers.

However, it does not return correct results when the contract holds assets.

As a consequence, this can be exploited by attackers to steal the contract's balance by triggering swaps with the `isWrapper` parameter set to `false`.

We classify this issue as major because the contract should not hold a balance, and this can only arise from incorrect usage, such as in the scenario mentioned where DEX swap functions enable users to perform arbitrary calls.

Recommendation

We recommend implementing logic that does not rely on the contract's absolute balance. Consider computing the return amount as the current balance minus the previous balance before the swap. This ensures to obtain the amount of tokens of the swap operation.

Status: Acknowledged

The client states that the contract is designed to be stateless and to not hold funds.

8. Missing emission of events for configuration changes

Severity: Minor

The functions `setAssetForwarder`, `setAssetBridge`, `setFlagToFactoryAdress`, `setFactorySetter` in `evm/src/dexspan/DexSpan.sol:215-240` and `setWNativeAddresses` in `evm/src/dexspan/DexSpan.sol:242` should emit events to ensure proper tracking and integration with off-chain bridge components like the forwarder.

However, they do not emit any events, leading to issues with off-chain bridge components.

We classify this issue as minor since only admins can execute the aforementioned functions.

Recommendation

We recommend emitting events and ideally implementing callback logic that ensures proper connection between the target contracts. For example, callback logic could prevent configuring the bridge as a forwarder or vice versa.

Status: Resolved

9. Missing addresses validation

Severity: Minor

In the following locations, addresses in function arguments are not validated:

- In `evm/src/dexspan/DexSpan.sol:204-214`, `assetForwarderAddress`, `native`, and `wrappedNative` are not validated.
- In `evm/src/dexspan/DexSpan.sol:215-219`, `_forwarder` is not validated.
- In `evm/src/dexspan/DexSpan.sol:221-225`, `_assetBridge` is not validated.
- In `evm/src/dexspan/DexSpan.sol:227-234`, `_factoryAddress` is not validated.

Recommendation

We recommend verifying that the addresses are not equal to `address(0)`.

Status: Resolved

10. Lack of contract ownership transfer functionality

Severity: Minor

In `evm/src/dexspan/DexSpan.sol`, the contract should provide a mechanism to update its owner.

Without such functionality, in case of issues with key management or a compromised owner, it is not possible to update the contract owner.

Recommendation

We recommend implementing a two-step contract ownership transfer mechanism.

Status: Partially Resolved

The client implemented a one-step contract ownership transfer mechanism.

11. Anomalies in hexadecimal flags' interval width can lead the transaction to revert

Severity: Minor

In `evm/src/dexspan/DexSpan.sol:527-552`, flags are defined as hexadecimal intervals.

However, the width of the interval is inconsistent and does not include some values.

Specifically, the first interval is `[0x001, 0x03E9)`, the second is `(0x03E9, 0x07D1)`, the third is `(0x07D1, 0x0BB9)`, and the last one is `[0x0BB9, 0x0FA1)`.

Consequently, the second and third intervals are comparatively smaller, and render the flags `0x03E9` and `0x07D1` unusable.

The global state variable mapping `flagToAddress` stores the address to swap with an external protocol related to the `uint256` defined flag. The address is then retrieved in:

1. `evm/src/dexspan/DexSpan.sol:573` `_swapOnUniswapV3`
2. `evm/src/dexspan/DexSpan.sol:602` `_swapOnOneInch`
3. `evm/src/dexspan/DexSpan.sol:626` `_swapOnParaswap`
4. `evm/src/dexspan/DexSpan.sol:655` `_swapOnExchangeInternal`

So it is possible to use a flag on a valid range that does not correspond to any real address, as the mapping only assigns one integer per address. This could lead to success on the range validation but reverts when trying to obtain the address. This code inconsistency would require assigning all individual integers ranges per corresponding address.

Recommendation

We recommend using one integer per address to avoid code inconsistencies and validation issues that result in unexpected reverts.

Status: Resolved

12. The swap function accepts ETH even if not defined in the token swap array leading to funds stuck in the contract

Severity: Minor

In `evm/src/dexspan/DexSpan.sol:439-449`, during the execution of the `_swapMultiInternal` function, if the user-defined `tokens` array does not contain ETH in the first position, and the transaction's `msg.amount` is greater than 0, the transaction should revert.

However, since there is no check in place, the transaction will not revert and excess ETH funds will be stuck in the contract.

We classify this issue as minor since the frontend is assumed to correctly construct messages.

Recommendation

We recommend adding a revert condition to prevent the user from inadvertently sending ETH to the contract.

Status: Resolved

13. The `assetBridge` and `assetForwarder` could drain funds from the contract by sending a faulty swap message

Severity: Minor

In `evm/src/dexspan/DexSpan.sol:404-420`, both the `assetForwarder` and the `assetBridge` addresses have the potential to siphon funds from the contract by invoking the `messageHandler` function with a failing swap message.

In fact, in the event of a swap failure, the `_amount` is transferred to the recipient without any validation or safeguards in place.

We classify this issue as minor since it can only be executed from the `assetBridge` and `assetForwarder` trusted addresses.

Recommendation

We recommend implementing checks and safeguards in the `messageHandler` function to prevent fund drainage in case of a failing message.

Status: Acknowledged

14. Insecure non-production contracts in the codebase

Severity: Minor

The `evm/src/dexspan/DAI.sol` and `evm/src/dexspan/REP.sol` contracts contain contracts for tokens with minimal specifications and a permissionless mint functionality.

We classify this issue as minor since these contracts are classified as testing/non-production code.

Recommendation

We recommend removing those files from the codebase or moving them to a testing-specific directory.

Status: Resolved

15. Missing role segregation between `DEFAULT_ADMIN_ROLE` and `FACTORY_SETTER_ROLE`

Severity: Minor

In `evm/src/dexspan/DexSpan.sol:236-241`, the system does not enforce segregation between the `DEFAULT_ADMIN_ROLE` and the `FACTORY_SETTER_ROLE` to maintain distinct administrative boundaries.

However, no such segregation is enforced, allowing the admin to potentially assume the responsibilities of the factory setter and vice versa.

As a consequence, this lack of role differentiation could lead to a concentration of power and responsibilities, undermining the security principle of least privilege.

Recommendation

We recommend maintaining separate accounts for each distinct role.

Status: Resolved

16. Avoid using `extcodesize` to check for externally owned accounts

Severity: Minor

The `isContract` function defined in `evm/src/dexspan/DexSpan.sol:669` might be used to check for externally owned accounts.

However, this might not work accurately if a contract does not have source code available during construction.

This means that while the constructor is running, it can make calls to other contracts, but `extcodesize` for its address returns zero as described in EXTCODESIZE Checks - Ethereum Smart Contract Best Practices.

Recommendation

We recommend removing the function completely.

Status: Acknowledged

17. Lack of input validation on `_swapMultiInternal` function parameters

Severity: Informational

The function `_swapMultiInternal` defined in `evm/src/dexspan/DexSpan.sol:477` iterates in line `498` all the elements provided by the caller in the `tokens` vector.

However, since for each of them, the execution accesses the `flags` and `dataTx` vectors at the same index, in case those vectors do not have the same length of the `tokens` vector an index out-of-bounds error will be raised causing the transaction to revert.

Recommendation

We recommend validating the length of the `flags` and `dataTx` arrays to be equal to the `tokens` array length before using them on the `_swapFloor` function invocation.

Status: Resolved

18. Missing UniswapV2 pair existence check

Severity: Informational

The internal function `_swapOnExchangeInternal` is meant to execute a swap using UniswapV2. It calls the factory to obtain the pool address for a given pair of tokens, but it does not check if the obtained address is different from zero.

Not checking if the pair exists results in an uncontrolled revert when calling `getReserves` from a zero address.

Recommendation

We recommend checking if the pair exists and reverting in a controlled way if it does not.

Status: Resolved

19. Functions called externally are defined as `public`

Severity: Informational

The `handleMessage`, `handleAssetBrigeMessage`, `setWNativeAddresses` `setAssetForwarder`, `setAssetBridge`, `setFlagToFactoryAdress`, and `setFactorySetter` functions defined in `evm/src/dexspan/DexSpan.sol` are called only externally, but defined as `public`, which is inefficient.

Recommendation

We recommend defining these contracts as `external`.

Status: Resolved

20. Use custom errors for gas efficiency

Severity: Informational

In several instances, throughout the codebase, the `if - revert()` and `require(something, "STRING")` patterns are used to catch errors.

Custom errors are more convenient, maintenance-friendly, and gas-efficient and therefore a best practice since Solidity v0.8.4 as explained in https://soliditylang.org/blog/2021/04/21/custom-errors/.

Affected instances are in:

1. `evm/src/dexspan/DexSpan.sol:231, 246, 247, 267, 401, 448, 473, 486, 525, 571, 574, 590, 600, 603, 614, 624, 627, 642, 656,` and `551`
2. `evm/src/dexspan/DexSpan.sol:44`
3. `evm/src/dexspan/libaries/Multicall.sol:23`
4. `evm/src/dexspan/libaries/TransferHelper.sol:14, 27,` and `41`

Recommendation

We recommend using only custom errors for error handling.

Status: Acknowledged

21. Unused functions, variables, parameters, and return values

Severity: Informational

The following functions, variables, parameters, and return values seem not to have any purpose in the codebase:

1. `isContract` in `evm/src/dexspan/DexSpan.sol:669`
2. `struct DexesArgs` in `evm/src/dexspan/DexSpan.sol:184`
3. Return value of the `setWNativeAddresses` function in `evm/src/dexspan/DexSpan.sol`
4. Return value of the `setFlagToFactoryAddress` function in `evm/src/dexspan/DexSpan.sol`
5. `bytes memory data` parameter in the `_swapOnUniswapV2` function in `evm/src/dexspan/DexSpan.sol`
6. `_linearInterpolation` function in `evm/src/dexspan/DexSpan.sol:106`

7. `_tokensEqual` function in `evm/src/dexspan/DexSpan.sol:116`

Recommendation

We recommend removing unused functions, variables, parameters, and return values.

Status: Acknowledged

22. Altered `_returnData` in Multicall contract

Severity: Informational

The implementation of the Mulitcall contract alters the return data. In `evm/src/dexspan/libraries/Multicall.sol:21-28` the _returnData becomes larger than it should be.

Although there are so far no security implications, altering the bytecode is against best practice and might lead to future errors.

Recommendation

We recommend using the battle-tested openzeppelin-contracts/contracts/utils/Multicall.sol library instead of a custom implementation.

Status: Resolved

23. Initializing variables to their default value is inefficient

Severity: Informational

Solidity assigns default values to declared variables. In the case of integers and unsigned integers, the default value is zero. For addresses, the default value is the zero address. For boolean values the default value is false.

The following lines of code contain variables initialized to their corresponding default value.

1. `evm/src/dexspan/DexSpan.sol:72`, `95`, `100`, `109`, and `142`
2. `evm/src/dexspan/DexSpanView.sol:41`, `64`, `69`, `78`, `111`, `222`, `223`, `232`, `241`, `291`, and `381`

Although in recent solidity compiler versions, this code pattern is optimized and removed, on older compiler versions this results in an extra gas cost performing an additional not required operation.

Recommendation

We recommend not initializing variables with their default value.

Status: Resolved

24. Use `++i` instead of `i++` to save gas

Severity: Informational

The operation `++i` costs less gas than `i++`, especially when used in for-loops.

However, the following locations implement the `i++` code pattern:

1. `evm/src/dexspan/DexSpan.sol:95`, `100`, `102`, `108`, `109`, `113`, `142`, and `454`
2. `evm/src/dexspan/DexSpanView.sol:64`, `69`, `71`, `77`, `78`, `82`, `111`, `223`, `232`, `241`, `242`, `291`, and `381`

Recommendation

We recommend changing the code to the `++i` pattern to save gas.

Status: Acknowledged

25. Contracts should implement a two-step role ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current role owners to execute a one-step role ownership transfer. While this is common practice, it presents a risk for the ownership of contract roles to become lost if the owner transfers a role to the incorrect address.

A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and execute the config update.

Recommendation

We recommend implementing a two-step role ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated.
2. The new owner account claims role ownership, which applies the configuration changes.

Status: Resolved

26. Code duplication for `handleMessage` and `handleAssetBridgeMessage`

Severity: Informational

In `evm/src/dexspan/DexSpan.sol:253-269`, the `handleMessage` and `handleAssetBridgeMessage` functions are identical except for the `require` statement which expects the caller to be respectively the `assetForwarder` or the `assetBridge`.

Recommendation

We recommend refactoring the mentioned functions into a single one with a unified require statement to remove duplicated code.

Status: Resolved

27. Duplicated computation leads to inefficiency

Severity: Informational

The `_swapMultiInternal` function in line `492` performs a subtraction between `userBalanceNew` and `userBalanceOld` to check if the returned amount is bigger than the `minReturn` amount.

If the check succeeds then the operation is repeated to store the value in the return variable leading to an unnecessary duplicated computation.

Recommendation

We recommend performing the operation once and checking the value before returning to save gas and avoid duplicated code.

Status: Resolved

28. `swapInSameChain` and `swapMultiWithRecipient` functions should emit different events

Severity: Informational

The function `swapInSameChain` calls the `swapMultiWithRecipient` function.

However, since both functions emit the same event, it will result in a duplicated event emission which is inefficient and potentially misleading.

Recommendation

We recommend creating different events for different operations and avoiding placing custom strings to differentiate them.

Status: Resolved

29. Use of magic numbers decreases maintainability

Severity: Informational

Throughout the codebase, hard-coded number literals without context or a description are used. Using such "magic numbers" goes against best practices as they reduce code readability and maintenance as developers are unable to easily understand their use and may make inconsistent changes across the codebase.

Instances of magic numbers are listed below:

- `evm/src/dexspan/DexSpan.sol:103`
- `evm/src/dexspan/DexSpan.sol:542-548`
- `evm/src/dexspan/DexSpanView.sol:351-352`
- `evm/src/dexspan/DexSpanView.sol:338`

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Acknowledged

30. Redundant parameter `minReturn` in `_swap` function

Severity: Informational

The `_swap` function, defined in `DexSpan.sol`, receives a `uint256` parameter named as `minReturn`.

This parameter is supposed to check that the swap returns the minimum amount specified by the user for the swap. However, when calling the `_swap` function from the `_swapFloor` function this parameter is always zero.

Moreover, the check for the minimum return amount is also performed on the caller functions, duplicating the checks.

Recommendation

We recommend removing unused parameters to save gas and avoid unnecessary code.

Status: Partially Resolved

31. Missing usage of the `IBridge` interface

Severity: Informational

In `evm/src/dexspan/DexSpan.sol:699-705`, the `assetBridge` variable is defined as `address`.

However, it should be explicitly defined as a variable of `IBridge` type.

Recommendation

We recommend changing the type of the `assetBridge` variable to `IBridge`.

Status: Acknowledged