PUBLIC

Code Assessment

of the Kernel Smart Contracts

December 2, 2024

Produced for

KERNEL

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help Kernel DAO with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Kernel according to Scope to support you in forming an opinion on their security risks.

Kernel DAO provides a token staking system that manages user assets within designated vaults. The contracts are upgradeable and form the basis for the development of a restaking protocol that Kernel DAO ultimately plans to implement.

The most critical subjects covered in our audit are integration with external protocols, DoS possibilities and functional correctness. The general subjects covered are upgradeability, gas efficiency and event emissions.

The most significant findings Broken clisBNB withdrawals and DoS by Donation have been corrected through code correction.

In summary, we find that the codebase provides a good level of security.

Moreover, we would like to highlight that it is necessary to make the assumptions described in Integration with Lista DAO to reliably integrate with the Lista DAO protocol. Failure to meet these assumptions could put Kernel user funds at risk.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	1
• Code Corrected	1
Medium-Severity Findings	0
Low-Severity Findings	4
• Code Corrected	4

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Kernel smart contract repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

Private Repository

V	Date	Commit Hash	Note
1	19 Nov 2024	2e6827284f1d7cac163764e9759c3192bf8b5f0f	v0.2 - Initial Version
2	27 Nov 2024	6c4ae16de4a6b39c3c368c825df9390356075c8f	v0.3 - Unsupport clisBNB
3	27 Nov 2024	f76f7acd0b06e136b1ab6783602606f71e929b1a	v0.4 - Base fixes
4	27 Nov 2024	560dc92fc7f63097c6c040da4703686889b756d4	v0.5 - Revised clisBNB
5	2 Dec 2024	dc57adb4ddf31a147461793c8e92dbcbcd573e13	v0.6 - DoS corrections

Public Repository

V	Date	Commit Hash	Note
1	19 Nov 2024	2e6827284f1d7cac163764e9759c3192bf8b5f0f	v0.2 - Initial Version
2	27 Nov 2024	6c4ae16de4a6b39c3c368c825df9390356075c8f	v0.3 - Unsupport clisBNB
3	27 Nov 2024	f76f7acd0b06e136b1ab6783602606f71e929b1a	v0.4 - Base fixes
4	27 Nov 2024	560dc92fc7f63097c6c040da4703686889b756d4	v0.5 - Revised clisBNB
5	2 Dec 2024	dc57adb4ddf31a147461793c8e92dbcbcd573e13	v0.6 - DoS corrections

For the solidity smart contracts, the compiler version `0.8.28` was chosen.

The files in scope were:

```
src/
    AssetRegistry.sol
    AssetRegistryStorage.sol
    HasConfigUpgradeable.sol
    KernelConfig.sol
    KernelConfigStorage.sol
    KernelVault.sol
    KernelVaultStorage.sol
    StakerGateway.sol
```

```
    StakerGatewayStorage.sol
    interfaces/
        IAssetRegistry.sol
        IHasConfigUpgradeable.sol
        IHasVersion.sol
        IHelioProvider.sol
        IKernelConfig.sol
        IKernelVault.sol
        IStakerGateway.sol
        IWBNB.sol
    libraries/
        AddressHelper.sol
```

2.1.1 Excluded from scope

All other files are out of scope. Other contracts of Kernel DAO are out-of-scope. Lista DAO contracts are out of scope. We limit the review of the integration according to our understanding and our assumption described in Integration with Lista DAO. We expect tokens to be regular BEP-20 tokens (no special behaviour). Also, see Supported Tokens. Also, see Roles and Trust Model.

2.2 System Overview

This system overview describes the initially received version (Version 5) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Kernel DAO implements Kernel which is a token staking system on BSC (Binance Smart Chain) that allocates assets to respective vaults. Currently, only deposits and withdrawals are supported without other special staking-related functionality. Note that Kernel DAO aims to use the implemented contracts as the foundation for future development of a comprehensive restaking protocol.

Kernel Config. The `KernelConfig` serves as a central data store for addresses and implements access control features as well as protocol-wide pausing options.

`KernelConfig.setAddress()` maps a name to an address. The function is callable by the `DEFAULT_ADMIN_ROLE`. Once a name has been assigned an address, the address for the name cannot be modified. The allowed names are listed below along with the expected values. Each address has a corresponding getter function to retrieve the address.

- `ASSET_REGISTRY`: The expected address is `AssetRegistry`.
- `CLIS_BNB`: The expected address is Lista DAO's clisBNB token.
- `HELIO_PROVIDER`: The expected address is Lista DAO's Helio Provider.
- `STAKER_GATEWAY`: The expected address is `StakerGateway`.
- `WBNB_CONTRACT`: The expected address is Wrapped BNB (WBNB).

Note that the getters may return `0x0` if the name has not been mapped.

`KernelConfig.pauseFunctionality()` and `KernelConfig.unpauseFunctionality()` pause and unpause a set of functions in the protocol. `ROLE_PAUSER` and `DEFAULT_ADMIN_ROLE` can pause and unpause, respectively. Only the functionality-sets below can be paused and unpaused.

- `VAULTS_DEPOSIT`: Defines whether vault deposits are paused (`KernelVault.deposit()`).

- VAULTS_WITHDRAW: Defines whether vault withdrawals are paused (KernelVault.withdraw()).
- PROTOCOL: Defines whether all functionality sets are paused. At the time of writing, that includes the deposits and withdrawals.

Getters for retrieving the pausing state are provided along with functions that allow enforcing that certain functionalities are not paused.

Similarly, access control can be enforced. Note that access control is inherited from AccessControlUpgradeable. For details and considerations, see Roles and Trust Model.

Asset Registry. The AssetRegistry serves as a central data store for mapping assets to vaults which correspond to KernelVault contracts.

AssetRegistry.addAsset() and AssetRegistry.removeAsset() add and remove support for an asset. More specifically, adding an asset takes a vault as an argument and maps the vault to its asset while tracking the asset. Removing an asset undoes the operation. There can only exist one vault per asset type. Note that respective getters are provided. Further, the asset registry wraps certain vault getters.

Kernel Vault. A KernelVault holds the tokens of an asset and keeps track of the users' balances.

KernelVault.setDepositLimit() sets a deposit limit for the vault. More specifically, this limits only new deposits. The function is only callable by ROLE_MANAGER. Note that the corresponding getter for retrieving the deposit limit is provided.

KernelVault.deposit() and KernelVault.withdraw() mint and burn the vault shares (1:1 to deposit/withdraw amounts; not a BEP-20 token). Both can only be called by the staker gateway and respect the corresponding pausing. More specifically, depositing computes a balance delta and mints the delta as shares to a given user. Withdrawing deducts the burned shares from the user and, if the staker gateway's operation requires, gives approval to the staker gateway. Additionally, the total balance of tokens deposited into a vault is tracked internally and kept in storage. Getters for the accounting, the vault balance and the ERC20 token balance are provided.

StakerGateway. The StakerGateway commands the KernelVault.deposit() and KernelVault.withdraw() functions to support staking and unstaking tokens.

Generally, staking and unstaking operations call KernelVault.deposit() and KernelVault.withdraw(). The functions below mainly differ in how the funds are moved to and from the vault due to differences in the staked assets. The staking functions available are the following:

- Regular BEP-20 tokens: StakerGateway.stake() moves the asset from the user to the vault. StakerGateway.unstake() withdraws from the vault and forces an approval so that the asset can be moved from the vault to the user.
- Native token (native BNB): StakerGateway.stakeNative() and StakerGateway.unstakeNative() perform the same operations as the corresponding functions for the BEP-20 tokens but additionally wrap and unwrap WBNB.
- Lista DAO's clisBNB: StakerGateway.stakeClisBNB() calls Lista DAO's Helio Provider's HelioProviderV2.provide() function to mint clisBNB to the vault with the native BNB provided and deposits to the vault. StakerGateway.unstakeClisBNB() calls HelioProviderV2.release() to burn the clisBNB and receive the underlying BNB in a potentially asynchronous manner.

Note that the following assumptions for Integration with Lista DAO must hold in order for users to retrieve their funds successfully.

2.2.1 Roles and Trust Model

The following roles are defined:

- ROLE_MANAGER: Partially trusted. Expected to set reasonable deposit limits.

- `ROLE_PAUSER`: Partially trusted. Expected to not pause the protocol unnecessarily.
- `ROLE_UPGRADER`. Fully trusted. Can upgrade contracts to steal funds. Could upgrade contracts to remove the `DEFAULT_ADMIN_ROLE` role's privileges and take over the governance.
- `DEFAULT_ADMIN_ROLE`: Fully trusted. Can set roles and unpause.

Other roles are:

- System contracts: Expected to be the reviewed contracts. Deviations might break the system.
- Tokens: Expected to be trusted and regular BEP-20 tokens (e.g. non-rebasing). Malfunctioning tokens may break the system for the given token.
- Lista DAO: Expected to work as documented. Malfunctioning might break the integration with Lista DAO.
- Users: Users are untrusted.

Note that the protocol is fully upgradeable.

2.2.2 Changes

In Version 2, the clisBNB integration was removed, resolving all notes and issues related to the integration.

In Version 3, all issues besides DoS by Donation are resolved.

In Version 4, the clisBNB integration was readded. Note that the issues removed in Version 2 have been resolved through code correction.

In Version 5, a total supply field has been added to the vaults to resolve DoS by Donation.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	1

- Broken clisBNB Withdrawals Code Corrected

Medium-Severity Findings	0
Low-Severity Findings	4

- DoS by Donation Code Corrected
- Incorrect Event Emissions Code Corrected
- Inefficient Array Operation Code Corrected
- Pause Getters Not Considering Protocol Pause Code Corrected

Informational Findings	3

- Lack of Events Code Corrected
- No Constants Used for Strings Code Corrected
- Style Choices Code Corrected

6.1 Broken clisBNB Withdrawals

Correctness High Version 1 Code Corrected

CS-KRNL-011

The clisBNB integration of the staker gateway breaks if insufficient funds are present in Lista DAO's master vault. Ultimately, users will not be able to unstake their funds.

For context, the unstaking process can be described as follows:

1. clisBNB is unstaked through the `StakerGateway`. The respective amount of Kernel vault shares are withdrawn.
2. `HelioProviderV2.release()` is called. This essentially burns the clisBNB and performs some other accounting actions on the `Interactions` contract. Then, the `MasterVault.withdrawETH()` function is called to unlock the capital in the master vault.
3. The master vault attempts to unlock funds according to the request. If the held BNB balance is insufficient, strategy withdrawals for the remaining amount will be performed to give sufficient funds to the account (staker gateway in this context).

Note that strategies not necessarily transfer the native token. Consider this strategy which has a delayed withdrawal due to staking requirements.

Hence, two issues arise:

1. The staker gateway cannot handle such behaviour. As a result users will not receive funds.

2. The returned amount by the `HelioProviderV2` will specify the amount unlocked, not the amount transferred. Thus, trying to transfer the returned `bnbAmount` will lead to reverts leading to a DoS of users.

Ultimately, the integration with clisBNB is broken and will lead to a loss of funds.

Code corrected:

Note that the clisBNB staking functionality was originally included in Version 1, temporarily removed in Version 2, and finally reintroduced in Version 4 with all issues from Version 1 resolved as described in the following.

Kernel DAO modifies the code so that unstaking clisBNB no longer specifies the `StakerGateway` as the recipient of funds on the Helio Provider. Instead, the final fund recipient is specified. As a consequence the user is now responsible for handling the pending withdrawals (stake withdrawals) as well as for receiving native token transfers from Lista DAO.

Note that the BNB amount emitted by the `AssetUnstaked` event in `unstakeClisBNB` may still be incorrect. Specifically, it will overvalue the unstaked amount in case Lista DAO's `mastervault` does not hold sufficient funds.

6.2 DoS by Donation

Correctness Low Version 1 Code Corrected

CS-KRNL-001

Two DoS possibilities exist due to the possibility to donate to the vault.

First, `AssetRegistry.removeAsset()` can be DoSed with donations. Namely, the zero-balance could never succeed if a user decides to donate a single wei to the vault.

Second, deposits can be DoSed. Namely, if funds are donated such that the held balance exceeds the deposit limit, all deposits will fail. Ultimately, that could be used as a gas griefing attack vector on users.

Note that for clisBNB, a DoS attack can be performed at no cost as the attacker can withdraw funds at any point in time. More specifically, this is possible through an external user delegating to the vault from their account via the Helio Provider.

Ultimately, important functionality could be DoSed.

Code corrected:

Kernel DAO resolves the issue by internally tracking the vault's total balance. Upon deposit, the deposit limit is checked against the internal accounting.

6.3 Incorrect Event Emissions

Correctness Low Version 1 Code Corrected

CS-KRNL-006

The `StakerGateway` emits `AssetStaked` when assets are staked. However, the emitted `AssetStaked.amount` is emitted inconsistently:

- `stakeClisBNB()`: Emits the minted vault shares which are equal to the balance delta in the vault.

- `stake()`: Emits the input `amount` but not the minted shares (balance delta in the vault). However, the values might not be equal, given that tokens with fees are supported.
- `stakeNative()`: Emits the `msg.value` which is equal to the balance delta.

Ultimately, for `stake()` the emitted amount could be wrong if tokens with fees are used. Similarly, the emitted amount could be wrong if during a callback tokens are additionally donated to the vault contract.

Note that, if for `AssetUnstaked` the emitted amount correspond to the amount to transfer, no issue arises. However, if the emitted amount should correspond to the burned shares, `unstakeClisBNB()` will emit a wrong amount.

Code corrected:

Note that the issue has been corrected. Namely, the vault now returns the deposited balance delta which is emitted in the event.

6.4 Inefficient Array Operation

Design | Low | Version 1 | Code Corrected

CS-KRNL-007

In `AssetRegistry`, pushing items to an array and removing items from it is performed in an extremely inefficient manner.

An asset is pushed as follows:

1. The full storage array is copied to memory. (`n+1` `SLOAD` operations)
2. A new memory array is created that has one more element.
3. The array from 1 is copied in memory to the array of 2. The new element is put into the last position.
4. The full new array is stored in storage. (`n+2` `SSTORE` operations)

This will already cost at least `2100 * n+1` gas. At 50 assets this will amount for over `100_000` gas. In contrast, `push()` could be used which would only need 2 storage writes which will cost less than `5000` gas.

Similarly, the removal of items is as inefficient and could be optimized by swapping the item to remove with the last item and calling `pop()` on the array.

Code corrected:

Now, regular `push` operations on array are used. Also, the removal has been optimized.

6.5 Pause Getters Not Considering Protocol Pause

Correctness | Low | Version 1 | Code Corrected

CS-KRNL-005

The functions `KernelConfig.isFunctionalityPaused()` does not consider whether the protocol is paused. Hence, if the protocol is paused, the getter will return `false`. Note that the `requireFunctionalityVault*()` functions consider the protocol-wide pausing.

Ultimately, the protocol pause should be considered in `isFunctionalityPaused()`, too.

Code corrected:

The code has been adjusted to support querying in two modes. Namely, users can specify whether protocol-wide pauses should be included.

6.6 Lack of Events

Informational | Version 1 | Code Corrected

CS-KRNL-008

Events can help off-chain information retrieve on-chain data more easily. However, some functions do not emit events appropriately. Below is a list of such functions:

1. `KernelVault.setDepositLimit()`: Deposit limit changes can not be retrieved by events.
2. `KernelConfig.pauseFunctionality()` and `KernelConfig.unpauseFunctionality()`: The set paused state including the paused functionality could be emitted.

Code corrected:

The events are now emitted.

6.7 No Constants Used for Strings

Informational | Version 1 | Code Corrected

CS-KRNL-009

Strings are always typed out. Typically, using constants for strings can make code less error-prone (e.g. typos). For example, `_getAddress("ASSET_REGISTRY");` could be replaced by an expression `_getAddress(STR_ASSET_REGISTRY);`.

Code corrected:

The code has been adjusted accordingly.

6.8 Style Choices

Informational | Version 1 | Code Corrected

CS-KRNL-010

1. `StakerGateway.getVault()` returns `address(_getVaultForAssetAddress(asset))` which includes double-casting. The internal function `_getVaultAddressForAssetAddress()` could be used directly. The current implementation leads to unnecessary nesting.
2. `StakerGateway._stake()` casts the `address asset` to `address`. Ultimately, the cast is redundant. Similarly, `KernelVault.getAsset()` casts an `address` to `address`.
3. Often, the custom errors encode a string with `string.concat`. For example, the `KernelVault.deposit()` could revert with signature `IKernelVault.DepositFailed`. In

one occurrence, integers are encoded to strings, making the code slightly more inefficient. Further, these error messages might be harder to parse. For example,

```
DepositLimitExceed(depositAmount, depositLimit, previousBalance)
```

could be easier to parse by front-ends compared to

```
DepositFailed(
    string.concat(
        "Unable to deposit an amount of ",
        Strings.toString(depositAmount),
        ": limit of ",
        Strings.toString(depositLimit),
        " exceeded"
    )
)
```

Code corrected:

The below list elaborates on which style changes have been applied.

1. Applied.
2. Applied.
3. Partially applied. Note that on several occasions the same pattern is used. Namely, that is for `IKernelConfig.NotStored`, `IKernelConfig.InvalidArgument`, `IKernelVault.DepositFailed`, `IKernelVault.WithdrawFailed`, `IStakerGateway.InvalidArgument` and `IStakerGateway.UnstakeFailed`. Note that for the usage of these, the way of reverting should be sufficient.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Floating Pragma

Informational | Version 1 | Acknowledged

CS-KRNL-002

The contracts use a floating solidity pragma: ^0.8.28. Contracts should be deployed with the same compiler version and flags that have been used during testing and audit. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Note that for interfaces a less restrictive version could be used so that third-parties can easily reuse the interfaces. Similarly, for libraries keeping the floating pragma could be useful if the library is designed to be used across a wider range of contracts.

Acknowledged:

Kernel DAO acknowledges the issue and does not fix the compiler version.

7.2 Gas Inefficiencies

Informational | Version 1 | Code Partially Corrected

CS-KRNL-003

Some operations consume more gas than necessary. Below is a non-exhaustive list of gas inefficiencies:

1. No `immutable` variables are used. The `KernelConfig` serves as a data store for important addresses. The mapping cannot be changed. Contracts such as the `StakerGateway` could leverage this immutability to retrieve the needed addresses on deployment of the implementation contract to reduce gas cost of executions. For example, `StakerGateway` would not need to always query the `KernelConfig`. Calls and storage reads could be saved.
2. Further, `HasConfigUpgradeable.configAddress` could be set as an `immutable`, given that the configuration address cannot change and is always the same.
3. `KernelVault.withdraw()` retrieves the staker gateway. However, given that `msg.sender` must be the staker gateway, `msg.sender` could be used to save gas.
4. In `KernelVault.deposit()` the current balance of the vault could be cached instead of making multiple calls to `_balance()`.

Code partially corrected:

Some optimizations have been applied. See the list below:

1. Not applied.
2. Not applied.
3. Applied. `msg.sender` is now used.

4. Applied. The value is now cached.

7.3 Pausing State Machine

Informational **Version 1** **Acknowledged**

CS-KRNL-004

The `KernelConfig.pauseFunctionality()` and the `KernelConfig.unpauseFunctionality()` do not validate whether the state is already paused and unpaused. Hence, pausing the same functionality twice in a row is possible. Ultimately, this is inconsistent with for example the asset registries adding and removing of assets where only added assets could be removed.

Acknowledged:

Kernel DAO acknowledges the issue but has decided not to make changes to the code.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Considerations for clisBNB Withdrawals

Note Version 4

To resolve Broken clisBNB withdrawals, the user is specified as the recipient of funds when redeeming clisBNB. Given that funds will potentially be staked, the responsibility of fully handling withdrawals is given to the user.

Hence, users should be aware that redeeming their clisBNB vault shares might lead to incomplete withdrawals in native tokens. As a consequence, users should monitor their pending withdrawals in Lista DAO's master vault.

8.2 Deposit Limit

Note Version 1

Users and governance should be aware that the deposit limit is enforced softly. Namely, the deposit limit only applies to new deposits. However, it could be violated if the deposit limit is reduced below the current supply.

8.3 Depositing to KernelVaults With Paused Withdrawal

Note Version 1

Users should be aware that contracts are pausable. Note that withdrawals can be paused while deposits could still be allowed. As a result, the deposited funds could be locked until the `DEFAULT_ADMIN_ROLE` unpauses withdrawals.

8.4 DoS by clisBNB Minimum Withdrawal Amounts

Note Version 1

Users depositing into the clisBNB vault could be DoSed by the minimum withdrawal amounts required by the Lista DAO contracts. Namely, the following check

```
require(
    amount >= minumumUnstake,
    "value must be greater than min unstake amount"
);
```

in the Helio Provider could trigger reverts for users.

However, unless Lista DAO sets very high minima, users should be able to top-up by depositing first. Interestingly, the deposits do not enforce any minimums.

Thus, governance and users should be aware of this and act accordingly.

8.5 Initializers

Note Version 1

The contracts are deployed by EOAs (no factories) and are expected to be initialized when deploying the proxies. Namely, that is expected to happen directly in the constructor of the proxies by passing the respective `_data`. See the ERC1997 implementation for reference.

8.6 Integration With Lista DAO

Note Version 1

The staker gateway integrates with Lista DAO's `HelioProviderV2` contract to enable users to deposit into a clisBNB Kernel vault. Below we give an overview of our understanding of ListaDAO. Given the lack of documentation, we limit the assessment according to that understanding.

Overview. The BNB provided to the Helio Provider will be deposited in the Lista DAO `masterVault` to follow yield earning strategies returning master vault shares (called `MasterVault.vaultToken` and `HelioProviderV2._ceToken`) that will be deposited as collateral to the Lista DAO CDP system as collateral on behalf of an account (through the `_dao` contract). In the context of the `StakerGateway`, that account will be `StakerGateway`, giving it the rights to claim the underlying collateral. Note that withdrawing is thus 1:1 with the deposited BNB (excluding fees).

Note that the yield generated is not distributed among the `MasterVault` depositors (and thus not among the `HelioProviderV2` depositors). However, collateral providers can be rewarded by airdrops and similar mechanisms. Such rewards can be delegated to another address.

clisBNB is a non-transferable token, representing such delegations. In the context of `StakerGateway`, the delegatee will be the respective vault.

Assumptions. For users of the clisBNB KernelVault to be able to correctly receive their share of BNB upon unstaking, we must assume the following:

1. The `ink` and `gem` balances generated through the `_dao` contract will always be sufficient to handle withdrawals. Ultimately, it is expected that a `_dao.deposit(stakerGateway, _ceToken, amount)` implies that `_dao.withdraw(stakerVault, _ceToken, amount)` is possible.
2. Similarly, the `_ceToken` escrowed in the respective gem join adapter should be sufficient to cover all other logic.
3. Ultimately, that implies that liquidations and any other mechanisms cannot impact what was deposited.
4. Accordingly, the balance of clisBNB tokens held by the clisBNB KernelVault cannot decrease unexpectedly.
5. It is expected that the MasterVault will return amounts accordingly. Note that there is no enforcement that strategies will return the expected amount of funds.

```
uint256 value = IBaseStrategy(strategy).withdraw(recipient, amount);
require(
    value <= amount,
```

```
    "invalid withdrawn amount"
);
```

For example, strategies could return zero. That is not expected and should never occur. `value` should always be as close as possible to the desired amount.

8.7 Rewards Unsupported

Note **Version 1**

According to Kernel DAO, clisBNB may be eligible for rewards and airdrops. Kernel DAO specified that not being able to retrieve such rewards immediately is intended and by design.

Note that future upgrades could change that.

8.8 StakerGateway May Receive BNB From Users

Note **Version 1**

The `enableNativeTokenReceive` modifier is used in order to ensure that the `StakerGateway` can only receive native BNB tokens from the WBNB contract. However, any user of the system may still transfer BNB to the staker gateway contract. Consider the options below:

1. During the low level call to `msg.sender` in `StakerGateway.unstakeNative()`, native token transfers are possible. BNB that is transferred to the staker gateway contract through this path can not be retrieved and will be stuck.
2. Similarly, with `selfdestruct` the gateway could receive funds.

Ultimately, the check is not always effective.

8.9 Supported Tokens

Note **Version 1**

Governance and users should be aware that not all tokens are supported. While generally, the protocol supports BEP-20 tokens, considerations are to be made for similar token types. Below is a (non-exhaustive) list of considerations:

1. Not all standard ERC-20 tokens are supported. Namely, ERC-20 (unlike BEP-20) does not enforce the existence of the `decimals()` function that is used during the initialization of vaults.
2. Tokens with fees deducting the fee amount from the sent amount are supported. However, tokens with fees topping up the transfer amount are unsupported.
3. Rebasing tokens are unsupported.
4. Reentrant tokens are generally supported (e.g. ERC-777). However, scenarios exist where reentrancy into governance functionality during `KernelVault.stake()` might lead to deposits to a vault that has been removed from the asset registry (stake and during a callback change the vault in the asset registry).
5. Tokens with blocklists might DoS a vault or its users for certain tokens.

Note that any future upgrade might affect the supported tokens.

8.10 Vault Removal

Note Version 1

An asset, and consequently a vault, can only be removed from the system if all users have fully withdrawn their funds, resulting in a `vault.totalBalance` of 0. This scenario is unlikely, meaning assets added to the staking system will likely remain supported indefinitely.