BAILSEC.IO

OFFICE@BAILSEC.IO

X: @BAILSECURITY

TG: @HELLOATBAILSEC

FINAL REPORT

Kernel

October 2024

Disclaimer:

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1. Project Details

Important:
Please ensure that the deployed contract matches the source-code of the last commit hash.

Project	Kernel
Website	kerneldao.com
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/Kelp-DAO/kernel-smart-contracts-private/tree/6a0a75fd89ecb51785cee78f951a31e8cfa87184
Resolution 1	https://github.com/Kelp-DAO/kernel-smart-contracts-private/tree/fd67ec221d0504ea6eea8bda5a87f05891c33a4a
Resolution 2	https://github.com/Kelp-DAO/kernel-smart-contracts-private/tree/48907573cb073dda15ae52c83e71e083031bf3b0/src

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1	1		
Medium	2	1		1
Low	2	1		1
Informational	10	7		3
Governance	1			1
Total	16	10		6

2.1 Detection Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior

3. Detection

Global

Issue_01	Governance Issue: Upgradeability
Severity	**Governance**
Description	Currently, governance of this contract has several privileges for invoking certain functions that can drastically alter the contract's behavior. This is possible due to the fact that all contracts are meant to be implementation contracts and used with a proxy.
Recommendations	Consider incorporating a Gnosis Multisignature contract as owner and ensuring that the Gnosis participants are trusted entities.
Comments / Resolution	Acknowledged.

AddressHelper.sol

The AddressHelper library is a simple utility library which is used throughout the architecture and executes address(0) checks.

The usage of this library is limited to the HasConfigUpgradeable and KernelConfig contracts.

Privileged Functions

- none

Issue_02	Usage of pushToFixedLengthAddressesArray and removeFromFixedLengthAddressesArray is unnecessary complex
Severity	**Low**
Description	Both aforementioned functions are used within the AssetRegistry to handle the assets array. More specifically, whenever an asset is added, it will be pushed to the array and whenever an asset is removed it is removed from the array. These functions are not manipulating the array via push / swap & pop but instead they are creating a completely new array for each operation. This is not only super gas inefficient but also introduces a lot of redundant code and complexity which can result in unexpected side effects.
Recommendations	Consider simply rewriting these functions, using push and swap & pop methods.
Comments / Resolution	Acknowledged.

KernelConfig/Storage

The KernelConfig contract is an externally deployed contract which corresponds to the configAddress described within the HasConfigUpgradeable contract. Any address with the admin role can determine the AssetRegistry, StakerGateway and WBNB address. After these addresses have been determined, they cannot be changed and will be used in the way described below.

Furthermore, this contract implements OpenZeppelins AccessControlUpgradeable contract for RBAC and allows for pausing and unpausing the whole protocol or only vault deposits / withdrawals. Pausing can be done by the manager role and unpausing can only be done by the admin role.

The following contracts are pointing towards this contract to fetch the current configurations:

- AssetRegistry:
  - Uses the config for access control purposes
    - onlyAdmin
    - onlyManager

- KernelVault:
  - Uses the config for access control purposes
    - onlyFromStakerGateway
    - onlyManager
  - Uses the config to check pausable states
    - onlyVaultsDepositNotPaused
    - onlyVaultsWithdrawNotPaused
  - Uses the config to fetch the StakerGateway address for approval purposes

- StakerGateway
  - Uses the config to fetch the WBNB address
  - Uses the config to fetch the AssetRegistry

Appendix: Core Invariants

INV 1: Only admins can set addresses

INV 2: Addresses can only be set once

INV 3: Addresses can only be set for the corresponding keys

INV 4: Only pausers can pause functionalities

INV 5: Only admins can unpause functionalities

Privileged Functions

- grantRole
- renounceRole
- setAddress
- unpauseFunctionality
- pauseFunctionality

Issue_03	One-time setting for addresses is error-prone
Severity	**Informational**
Description	The one-time setting for AssetRegistry, StakerGateway and WBNB ensures immutability for these variables but can also result in issues if they are accidentally set incorrect, as this would require a new deployment: *// check previous address is empty (address can be set only once)* *require(* *addresses[k] == address(0),* *InvalidArgument(string.concat("Setting address for key ", key, " is permitted only once"))* *);*
Recommendations	Consider if this can be accepted, if not, consider removing this check, which would however introduce a governance issue.
Comments / Resolution	Acknowledged.

Issue_04	Unused imports
Severity	**Informational**

Description	The contract contains one or more imports which are completely unused. This will increase contract size for no reason and can confuse third-party reviewers:
	
	import { IAssetRegistry } from "src/interfaces/IAssetRegistry.sol";
	import { IStakerGateway } from "src/interfaces/IStakerGateway.sol";

Recommendations	Consider removing these unused imports.
Comments / Resolution	Resolved.

HasConfigUpgradeable

The HasConfigUpgradeable contract is a helper contract which is inherited by the following contracts:

- AssetRegistry
- KernelVault
- StakerGateway

Upon initialization, the configAddress variable is set which points to the KernelConfig contract. The only purpose of this contract is to expose the correct configAddress which is fetched upon multiple operations, as explained within the KernelConfig/Storage section.

Appendix: Core Invariants:

INV 1: Child contracts must call __HasConfig_init

Privileged Functions

- none

No issues found

AssetRegistry/Storage

The AssetRegistry contract is a simple, externally deployed registry contract that keeps track of the assetToVault mappings. Addresses with the manager role are allowed to add a new vault for an asset and addresses with the admin role can remove it again. The RBAC from the KernelConfig contract is used for this purpose.

Each asset can have only one linked vault at a time and the assetToVault mapping is used for vault deposits and withdrawals via the StakerGateway contract.

The AssetRegistry is invoked within multiple operations of the StakerGateway contract, to determine the corresponding vault for an asset.

Appendix: Core Invariants

INV 1: Only managers can add assets

INV 2: Only admins can remove assets

INV 3: Assets cannot be added twice

INV 4: Vault must have zero balance before removal

Privileged Functions

- removeAsset
- addAsset

Issue_05	Manager can permanently add wrong vault to an asset
Severity	**High**
Description	Contrary to the admin role, the manager role is likely a less trusted role (which is inherently indicated by introducing a second role alongside the admin role). The manager role can permanently add a wrong vault to an asset which breaks the architecture for adding the correct vault to a token. Even worse is that this can result in stealing tokens from users. Consider the example where it is intended to add a vault for USDT, the manager can execute the following steps to prevent the addition of the correct vault and steal deposited tokens: a) The Kernel team announces the addition of a USDT vault b) The address with the manager role frontruns the legit addAsset call as follows: -> deploy a malicious vault which exposes the exact same interface as the KernelVault but additionally exposes a rug() function which allows to withdraw all tokens -> call addAsset which maps the asset to the vault -> transfer 1 WEI USDT to the vault to prevent the removal of the vault (or even design Vault.balance() to never return zero) c) If users now deposit via the StakerGateway contract, assets are being transferred to the malicious vault and can be stolen by the manager. Optionally, even during the normal business logic, any address with the Manager role can just regularly deploy a slightly modified KernelVault contract which exposes a backdoor, which would likely not even stand out until the plug is pulled.

Recommendations	Consider only allowing addresses with the admin role to add new vaults.
Comments / Resolution	Resolved, only the admin can add new vaults.

Issue_06	Malicious user can DoS removeAsset via token donation
Severity	**Medium**
Description	The removeAsset function allows to remove the assetToVault linkage, whenever the balance of the corresponding vault is zero: *// check vault has no deposits* *address vault = assetToVault[asset];* *require(IKernelVault(vault).balance() == 0, VaultNotEmpty());* A malicious user can permanently prevent the removal of vaults by transferring 1 WEI to the vault.
Recommendations	Consider making this requirement optional.
Comments / Resolution	Acknowledged.

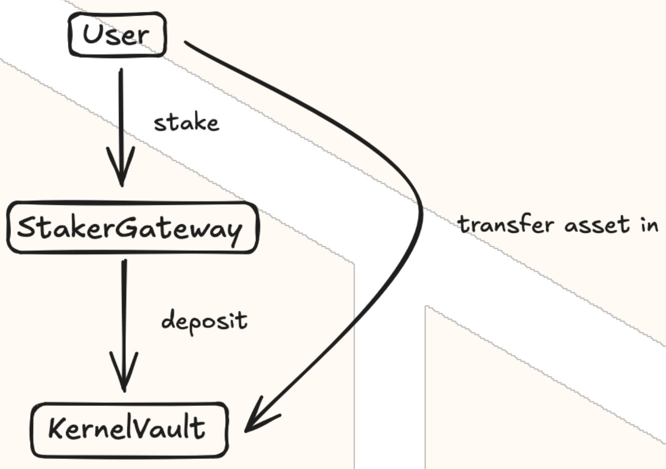
StakerGateway/Storage

The StakerGateway contract forms the entry contract for users to deposit different assets into different KernelVaults (as per AssetRegistry settings). It can be considered as a router which does not hold any important state and only interacts as an intermediate contract to assign the asset to its corresponding vault.

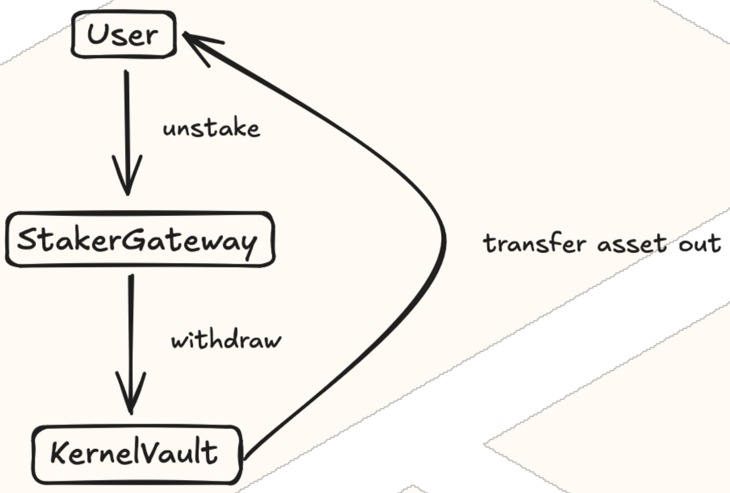
Besides of the standard stake and unstake logic, it incorporates logic for staking and unstaking the native BNB token via stakeNative and unstakeNative by simply allowing users to provide BNB as msg.value, wrapping it to WBNB and depositing it into the Kernel. For unstaking it will simply unwrap WBNB to BNB and transfer it to the user via a low-level call.

The staking and unstaking flow can be illustrated as follows:

a) **stake:**



b) **unstake:**



Appendix: Core Invariants:

INV 1: The contract must only accept BNB from the WBNB contract during withdrawals

INV 2: The contract must fetch the valid vault for a corresponding asset

Privileged Functions

- none

Issue_07	Accidentally transferred BNB is lost
Severity	**Low**
Description	The receive function was implemented in an effort to accept the BNB which is received during the WBNB.withdraw call within the unstakeNative function. However, currently there is no limitation in it which means that anyone can accidentally transfer BNB towards this contract, which then means the BNB will be permanently stuck.
Recommendations	Consider ensuring that only the WBNB address can transfer BNB to this contract. It must furthermore be ensured that the logic within the fallback function does not consume more than the 2300 provided gas by the .transfer call from the WBNB contract, as otherwise the transaction runs out of gas. (https://bailsec.io/tpost/58panskns1-ether-transfers-in-solidity-transfer-sen)
Comments / Resolution	Resolved.

Issue_08	Lack of zero check for amount / msg.value
Severity	**Informational**
Description	Currently, the stake and stakeNative function do not check for zero amounts. While we could not identify an issue out of that, most of the time, exploits happen due to arbitrary user inputs or users invoking functions which are not meant to be invoked by users, one can argue that a large user flexibility is a great seed for exploits. Therefore, at BailSec, we are of the opinion that codebases should never provide more user flexibility than necessary during the normal business logic.
Recommendations	Consider validating msg.value and the amount parameter to not be zero during stake and stakeNative.
Comments / Resolution	Resolved, such a check has been implemented.

Issue_09	Contract does not support transfer-tax tokens and other non standard tokens
Severity	**Informational**
Description	This contract is not compatible with transfer-tax tokens. If these token types are used for any purpose within the contract, this will result in down-stream issues and inherently break the accounting. This issue can be considered existent in the KernelVault contract as well. This counts furthermore for all weird token implementations (rebasing, debasing, etc)
Recommendations	Consider not implementing these tokens and consider conducting thorough DD before adding new tokens. Bailsec can offer such a DD service.
Comments / Resolution	Resolved, the vault balance is cached before the deposit which now allows to exactly calculate how much the vault has received.

KernelVault/Storage

The KernelVault contract is the main contract which holds funds within the Kernel architecture. Each KernelVault contract is tied to its own ERC20 token and users can deposit and withdraw their tokens via interacting with the StakerGateway contract which then simply transfers and pulls funds to each corresponding KernelVault contract.

The KernelVault contract simply stores deposited funds and keeps an accounting system which keeps track of how much tokens which address has deposited.

Furthermore, this contract employs a depositLimit which enforces an upper limit of how much tokens can be deposited to a KernelVault.

Appendix: Deposit Limit

Upon each deposit, it is ensured that the incoming deposit does not exceed the current determined depositLimit. The depositLimit can further be changed by the manager role without any limitations.

Appendix: Core Invariants:

INV 1: Only the StakerGateway contract can invoke the deposit and withdraw functions

INV 2: The total amount in the vault cannot exceed the specified deposit limit

INV 3: After a deposit, the user's balance is increased by the deposited amount

INV 4: After a withdrawal, the user's balance is decreased by the withdrawn amount

INV 5: Users cannot withdraw more than their current balance

Privileged Functions

- grantRole
- revokeRole
- setDepositLimit

Issue_10	Blunder within depositLimit check falsifies the limit
Severity	**Medium**

Description	During the deposit function, the depositLimit ensures that users can never deposit more tokens than the actual limit:
	
	`// check if deposit limit is exceeded`
	`require(`
	`amount + _balance() <= depositLimit,`
	`DepositFailed(`
	`    string.concat(`
	`    "Unable to deposit an amount of ",`
	`    Strings.toString(amount),`
	`    ": limit of ",`
	`    Strings.toString(depositLimit),`
	`    " exceeded"`
	`    )`
	`)`
	`);`
	
	This check is incorrect as the tokens have already been transferred in and thus are reflected within _balance():
	
	`/**`
	`* @notice Returns the Vault's balance of the managed asset`
	`*/`
	`function _balance() private view returns (uint256) {`
	`return IERC20(asset).balanceOf(address(this));`

	`}` However, due to the fact that the amount is added on top of it, the provided deposit amount for the check is basically twice as high as in reality.
Recommendations	Consider simply removing the addition of the amount parameter when checking for the depositLimit. Moreover, it can make sense to implement internal accounting for total deposits which fixes a bug where users accidentally transfer tokens to this contract, this is however not necessarily needed as in such a case governance can simply offset the locked tokens by increasing the depositLimit. Furthermore, no one would intentionally execute such an attack as this means the attacker loses tokens for nothing in return.
Comments / Resolution	Resolved, the balance post-transfer is now checked against the depositLimit.

Issue_11	Redundant uint8.max check
Severity	**Informational**
Description	The _getAssetDecimals function fetches the decimals from the corresponding ERC20 token as follows: *uint8 returnedDecimals = ERC20(assetAddr).decimals();* Afterwards, the following check is executed: *// check if returnet value is valid require(returnedDecimals <= type(uint8).max, InvalidArgument(string.concat("Invalid number of decimals returned by asset ", Strings.toHexString(assetAddr))));* This check is redundant as the previous call would already revert if uint8.max is exceeded.
Recommendations	Consider removing the redundant check.
Comments / Resolution	Resolved, this function has been removed and the decimal fetching is simplified.

Issue_12	Redundant approval practice
Severity	**Informational**
Description	The withdraw function approves the StakerGateway contract to consume the corresponding asset to honor the user withdrawal. This is done as follows: *// update allowance of StakerGateway* *SafeERC20.safeDecreaseAllowance(IERC20(asset), stakerGateway, 0);* *SafeERC20.safeIncreaseAllowance(IERC20(asset), stakerGateway, amount);* This practice is redundant as safeDecreaseAllowance call does nothing.
Recommendations	Consider simply executing a forceApprove call.
Comments / Resolution	Resolved, this has been removed.

Issue_13	Default depositLimit of zero prevents deposits until manager sets a limit
Severity	**Informational**
Description	The contract exposes a depositLimit which exposes an upper limit of how much tokens users can deposit into the contract. This limit is zero after deployment which means that the manager needs to set it before users can deposit tokens.
Recommendations	Consider if that is by design choice to prevent users from depositing before it is set. If not, consider setting it during the contract initialization.
Comments / Resolution	Acknowledged.

Issue_14	Missing address(0) check for assetAddr
Severity	**Informational**
Description	The initialize function allows governance to initially set the assetAddr and configAddr. There is currently no address(0) check for the assetAddr.
Recommendations	Consider executing an address(0) check for assetAddr.
Comments / Resolution	Resolved.

Issue_15	Redundant usage of ERC20 import
Severity	**Informational**
Description	The contract imports the ERC20 contract: *import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";* which is then used for the following purpose: *uint8 returnedDecimals = ERC20(assetAddr).decimals();* This practice is redundant as also the IERC20 interface can be used for this purpose.
Recommendations	Consider using IERC20 and remove the ERC20 import.
Comments / Resolution	Resolved.

Issue_16	_balance check against depositLimit may become inaccurate if tokens are accidentally being donated
Severity	**Informational**
Description	Currently, the depositLimit check is enforced as follows: *require(* *_balance() <= depositLimit,* *DepositFailed(* *string.concat(* *"Unable to deposit an amount of ",* *Strings.toString(depositAmount),* *": limit of ",* *Strings.toString(depositLimit),* *" exceeded"* *)* *)* *);* This check may become inaccurate if accidentally users transfer tokens to the KernelVault contract.
Recommendations	In such a scenario, consider simply increasing the depositLimit by the amount of stuck tokens.
Comments / Resolution	Acknowledged.