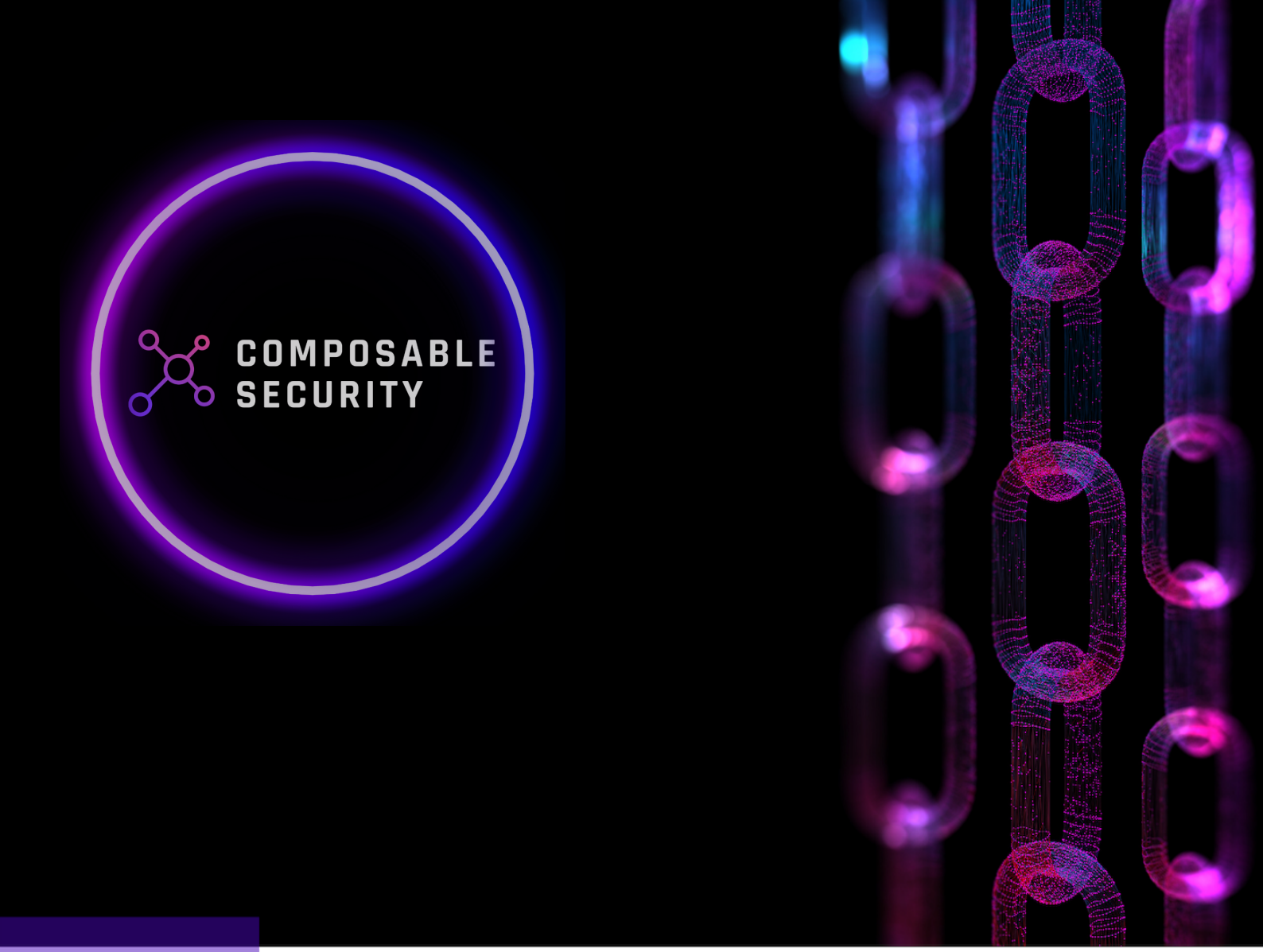


COMPOSABLE SECURITY

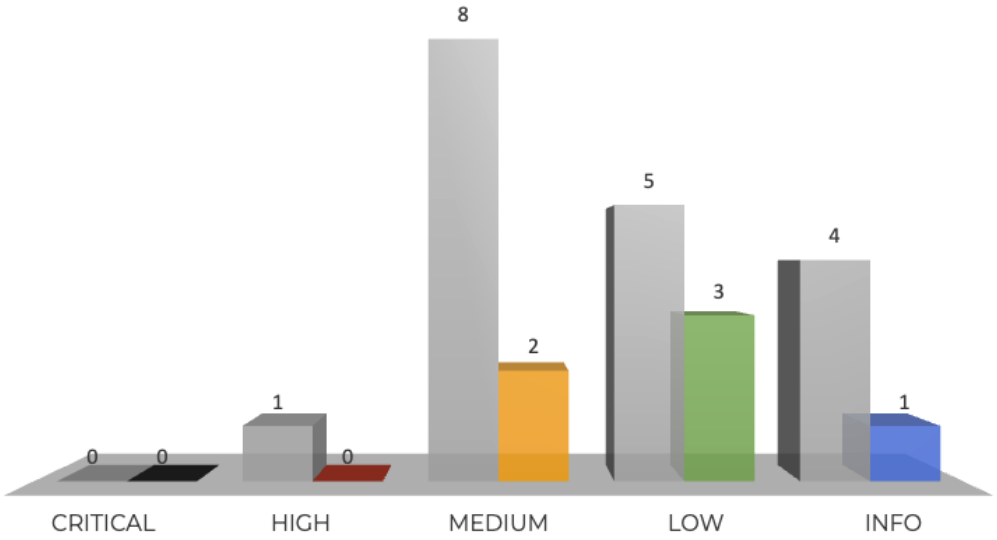
REPORT

Smart contract security review for YieldNest

Prepared by: Composable Security
Report ID: YINE-c7850b7c
Test time period: 2024-12-13 - 2024-12-20
Retest time period: 2025-01-02 - 2025-01-06
Report date: 2025-01-06
Version: 1.0
Visit: composable-security.com

Contents

1. Retest summary (2025-01-06)



The description of the current status for each retested vulnerability and recommendation has been added in its section.

1.1. Results

The **Composable Security** team was involved in a one-time iteration of verification whether the vulnerabilities detected during the tests (between 2024-12-13 and 2024-12-20) were removed correctly and no longer appear in the code.

The current status of detected issues is as follows:

- 1 **high** vulnerability has been removed from the code.
- 8 vulnerabilities with a **medium** impact on risk were handled as follows:
  - 6 have been fixed,
  - 2 have been acknowledged.
- 5 vulnerabilities with a **low** impact on risk were handled as follows:
  - 2 have been fixed,
  - 3 have been acknowledged.
- 4 security **recommendations** were handled as follows:
  - 3 have been implemented,
  - 1 have been acknowledged.

Even though the team hasn't fixed all the medium issues in the current iteration due to the limited risk of exploitation, they are aware of them and plan to make improvements.

In addition to removing the identified bugs, the contracts were refactored by the team.

1.2. Scope

The retest scope included the same contracts, on a different commit in the same repository.

GitHub repository: https://github.com/yieldnest/yieldnest-vault
CommitID: a9764a5b7e489f10b92501cd66acb4fa05c347fe

GitHub repository: https://github.com/yieldnest/yieldnest-vault/tree/bnb-max-vault
CommitID: 74d53eccc291e1fcced590b40dcff62e2ad98653

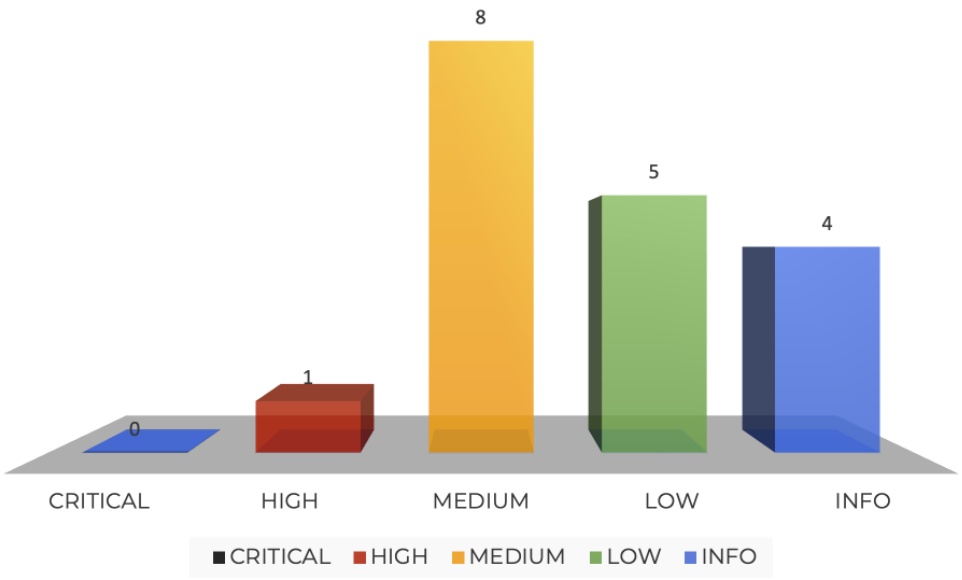
GitHub repository: https://github.com/yieldnest/yieldnest-kernel-lrt
CommitID: 62f2505cfca74ed590e27c59c921cb72d4297a68

2. Current findings status

ID	Severity	Vulnerability	Status
YINE-c7850b7c-H01	HIGH	Invalid decimals used to calculate total assets	FIXED
YINE-c7850b7c-M01	MEDIUM	Inflating vault's share rate via reentrancy	FIXED
YINE-c7850b7c-M02	MEDIUM	Invalid amounts of asset returned by strategies	FIXED
YINE-c7850b7c-M03	MEDIUM	Fee on withdrawal from strategy can lead to protocol's loss	ACKNOWLEDGED
YINE-c7850b7c-M04	MEDIUM	Inability to unstake required assets from Kernel	ACKNOWLEDGED
YINE-c7850b7c-M05	MEDIUM	Rate provider for ynBNBx is missing ynClisBNBk strategy	FIXED
YINE-c7850b7c-M06	MEDIUM	Invalid rate for YieldNest vaults	FIXED
YINE-c7850b7c-M07	MEDIUM	Vault not compliant with ERC4626 in maxWithdraw function	FIXED
YINE-c7850b7c-M08	MEDIUM	Back-running the rewards transfer	FIXED
YINE-c7850b7c-L01	LOW	Ability to withdraw non-active assets	FIXED
YINE-c7850b7c-L02	LOW	Decimals should be read from asset	ACKNOWLEDGED
YINE-c7850b7c-L03	LOW	Lack of clear limitations on fees	ACKNOWLEDGED
YINE-c7850b7c-L04	LOW	Inability to deposit and withdraw different asset than the base	ACKNOWLEDGED
YINE-c7850b7c-L05	LOW	Invalid modifier in KernelClisStrategy	FIXED

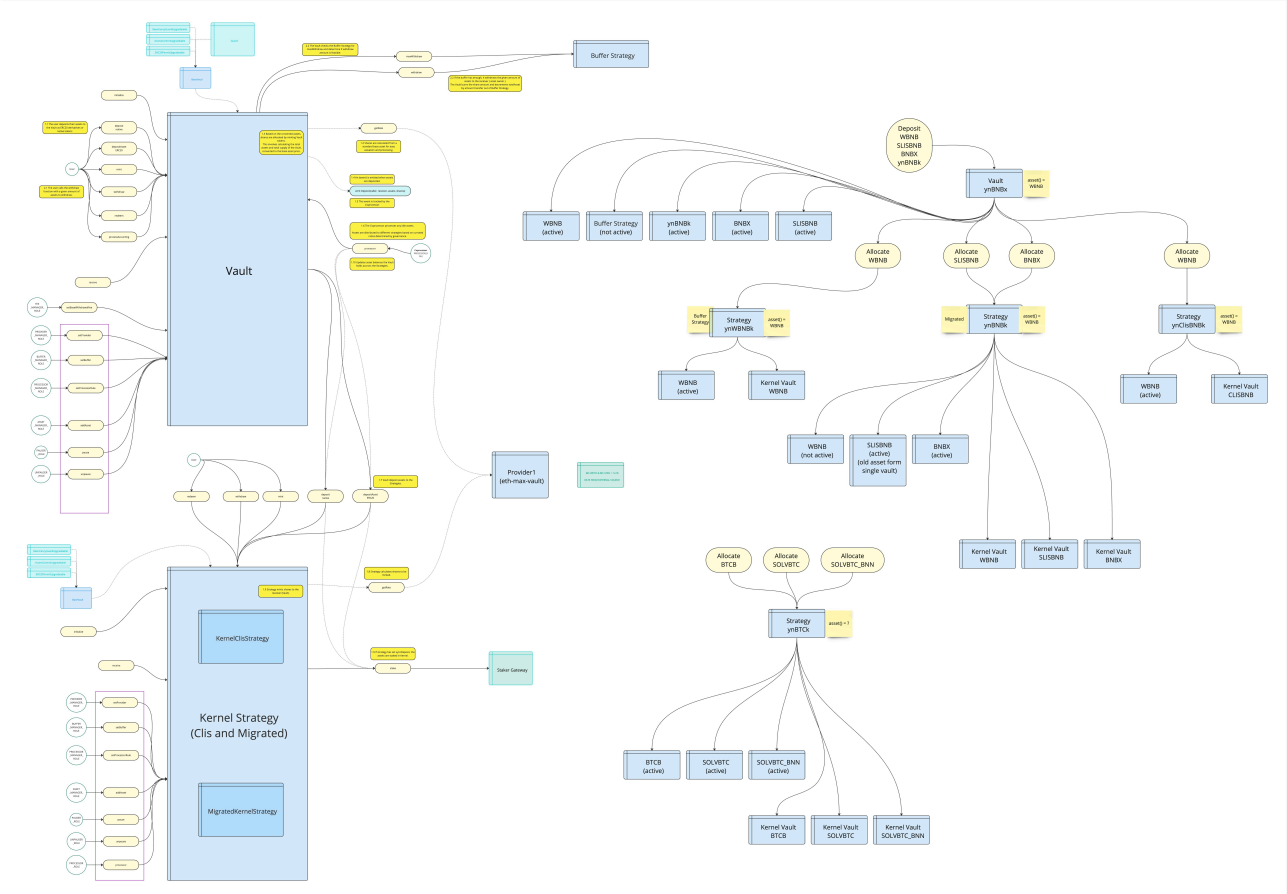
ID	Severity	Recommendation	Status
YINE-c7850b7c-R01	INFO	Remove hardcoded rate for SolvBTC.BBN	IMPLEMENTED
YINE-c7850b7c-R02	INFO	Remove unnecessary code	IMPLEMENTED
YINE-c7850b7c-R03	INFO	Use getPrices instead of getCurveEmaEthPerFrxEth	ACKNOWLEDGED
YINE-c7850b7c-R04	INFO	Emit events for important state changes	IMPLEMENTED

3. Security review summary (2024-12-20)



3.1. Client project

The project focuses on YieldNest integration with Kernel on BNB Chain and the newest Max Vault implementation.



The integration enables managing yield strategies on the BNB chain using the Kernel protocol. This partnership aims to integrate Kernel's infrastructure with YieldNest's Liquid Restaking Tokens (LRTs), specifically ynBNBk and ynBTCk.

- **ynBNBk:** This LRT combines existing ynBNB - a liquid restaking token on BNB Chain with Kernel's restaking infrastructure to optimize yield generation.
- **ynBTCk:** This LRT enables users to restake their Bitcoin assets on the BNB Chain, leveraging Kernel's infrastructure for enhanced returns.

The Max Vault Architecture is designed to give users streamlined access to yield opportunities across multiple DeFi protocols. The system design involves ERC4626-compatible Vaults (deposit vaults) and downstream Strategies that handle further allocation of assets to e.g. restaking connectors.

Architecture consinsts of the following key components

- **Vault (ERC4626)**
  - Core deposit contract that converts user deposits into a standardized share token.
  - Manages accounting (e.g., total assets, total shares).
  - Accepts multiple derivative or native assets but denominates shares in a single, major base asset (e.g., ETH, BTC, USD).
- **Strategies**
  - Downstream contracts that allocate assets into DeFi protocols (Kernel, etc.).
  - Receive allocations from the Vault and generate yield.
  - Return capital when users withdraw or when the Vault rebalances.
- **Coprocessor**
  - Monitors the Vault's deposit and withdraw events.
  - Calls `processor` function to move assets to/from Strategies.
  - Makes sure everything stays in sync without unnecessary gas overhead.
- **Rate Provider**
  - Feeds the Vault with current price/conversion data for all supported assets.
- **Buffer Strategy**
  - Strategy that holds enough liquidity for immediate withdrawals.
  - Prevents the need to exit more complex or illiquid Strategies for everyday withdrawal requests.

3.2. Results

The **YieldNest** engaged Composable Security to review security of **Integration with Kernel on BNB Chain**. Composable Security conducted this assessment over 1 week with 2 engineers.

The summary of findings is as follows:

- 1 vulnerability with a **high** impact on risk was identified. The potential consequence is:
  - The `totalAssets` variable will be inaccurately calculated, leading to theft of assets or potential losses for users due to undervalued `totalAssets`.
- 8 vulnerabilities with a **medium** impact on risk were identified.
- 5 vulnerabilities with a **low** impact on risk were identified.
- 4 **recommendations** have been proposed that can improve overall security and help implement best practice.
- The most important issues detected concern `totalAsset` accounting and fee mechanisms.
- The YieldNest team was very involved and actively participated in the testing of the protocol. They independently identified a number of issues detailed in the audit report.
- At the time of audit, the code was still in development. Subsequent corrections and enhancements were made.
- The protocol assumes that the decimal precision returned by the rate provider matches the decimals of the base asset. This should be carefully verified when integrating new rate providers.
- Max Vaults accept multiple assets, all denominated in one base asset (such as WBNB, WETH, or WBTC). However, withdrawals always occur in the base asset. As a result, the overall security of the protocol is only as strong as its weakest integrated asset. If one asset becomes compromised or loses its peg, it could enable attackers to drain the base asset reserves through that vulnerable asset.
- The YieldNest vault implement fee on withdrawals that is not transferred out and kept in the vaults. It means that the fee is included in `totalAssets`, increasing the share value and cannot be withdrawn by the team. This is intended behavior, as the team does not plan to collect fees. However, it is important to not that withdrawal of more than 50% of assets can create an arbitrage opportunity, because the share price would increase more than the withdrawal fee.
- Since the project relies on multiple integrations, it is crucial to maintain ongoing communication with partner projects and to closely monitor their updates and changes.

Composable Security recommends that **YieldNest** complete the following:

- Address all reported issues.
- Extend unit tests with scenarios that cover detected vulnerabilities where possible.
- Develop an internal checklist that includes identified vulnerabilities to help prevent similar issues during future code reviews and development cycles.
- Consider whether the detected vulnerabilities may exist in other places (or ongoing projects) that have not been detected during engagement.
- Secure privileged roles private keys as advised in Secure Private Key Management for DApps

3.3. Centralization Risk

The current system implementation lacks full decentralization, which permits critical operations—such as adding arbitrary assets and executing external calls on behalf of the vault—to be carried out by individuals holding specific predefined roles.

To address the centralization risk associated with the protocol, several mitigative measures have been put in place:

- **Permissions granularization**: Each critical operation is restricted to the holder of a designated role, thereby limiting access.
- **External calls guarding**: Only select external calls to approved targets are permitted, enhancing security.
- **Timelocks**: Critical changes are subject to timelocks, allowing users time to respond and mitigate potential threats arising from private-key exposure.
- **Multisigs**: Operations that are timelocked require execution by multisignature wallets, adding an additional layer of security.

3.4. Scope

The scope of the tests included selected contracts from the following repositories.

GitHub repository: https://github.com/yieldnest/yieldnest-vault
CommitID: c7850b7c65443e198872ec6f08fb8f11a493496a

GitHub repository: https://github.com/yieldnest/yieldnest-vault/tree/bnb-max-vault
CommitID: 9cd8c99fb0a40cfb2e89dbbd7b8fada4106bc577

GitHub repository: https://github.com/yieldnest/yieldnest-kernel-lrt
CommitID: 46b99aa3f5b519be0d30eb37bf44e669f34b4620

The detailed scope of tests can be found in Agreed scope of tests.

4. Project details

4.1. Projects goal

The Composable Security team focused during this audit on the following:

- Perform a tailored threat analysis.
- Ensure that smart contract code is written according to security best practices.
- Identify security issues and potential threats both for **YieldNest** and their users.
- The secondary goal is to improve code clarity and optimize code where possible.

4.2. Agreed scope of tests

The subjects of the test were selected contracts from the **YieldNest** repository.

GitHub repository: https://github.com/yieldnest/yieldnest-vault/tree/eth-max-vault
CommitID: c7850b7c65443e198872ec6f08fb8f11a493496a

Files in scope:

```
.
├── BaseVault.sol
├── Vault.sol
├── ynETHxVault.sol
└── module
    ├── FeeMath.sol
    ├── Provider.sol
    └── Guard.sol
```

GitHub repository: https://github.com/yieldnest/yieldnest-vault/tree/bnb-max-vault
CommitID: 9cd8c99fb0a40cfb2e89dbbd7b8fada4106bc577

Files in scope:

```
.
└── module
    └── Provider.sol
```

GitHub repository: https://github.com/yieldnest/yieldnest-kernel-lrt
CommitID: 46b99aa3f5b519be0d30eb37bf44e669f34b4620

Files in scope:

```
.
```

```
├── KernelClisStrategy.sol
├── KernelStrategy.sol
├── MigratedKernelStrategy.sol
└── module
    ├── BNBRateProvider.sol
    ├── BTCRateProvider.sol
    └── BaseKernelRateProvider.sol
```

Documentation:

- Max Vault Architecture
- Buffer Strategy

4.3. Threat analysis

This section summarizes the potential threats that were identified during initial threat modeling performed before the audit. The tests were focused, but not limited to, finding security issues that could be exploited to achieve these threats.

Key assets that require protection:

- Assets (deposited tokens)
- Exchange rate
- Protocol availability
- Private keys of the owner and privileged roles

Potential attackers goals:

- Theft of user's assets.
- Lock users' funds in the contract.
- Collecting more rewards than other users.
- Preventing other users from receiving rewards.
- Reducing the yield of the protocol.
- Preventing withdrawals.
- Brick the contract, so that others cannot use it.

Potential scenarios to achieve the indicated attacker's goals:

- Incorrect integration with Kernel protocol.
- Theft of funds stored in the strategy vault.
- Shares manipulation with donation attack.
- Manipulation of asset's exchange rate.
- Unauthorized minting of tokens.
- Lack of compliance with ERC4626.
- Influence or bypass the business logic of the system.

- Privilege escalation through incorrect access control to functions or badly written modifiers.
- Design issues.
- Excessive power, too much in relation to the declared one.

4.4. Testing methodology

Smart contract security review was performed using the following methods:

- Q&A sessions with the **YieldNest** development team to thoroughly understand intentions and assumptions of the project.
- Initial threat modeling to identify key areas and focus on covering the most relevant scenarios based on real threats.
- Automatic tests using slither.
- Custom scripts (e.g. unit tests) to verify scenarios from initial threat modeling.
- **Manual review of the code.**

4.5. Disclaimer

Smart contract security review **IS NOT A SECURITY WARRANTY**.

During the tests, the Composable Security team makes every effort to detect any occurring problems and help to address them. However, it is not allowed to treat the report as a security certificate and assume that the project does not contain any vulnerabilities. Securing smart contract platforms is a multi-stage process, starting from threat modeling, through development based on best practices, security reviews and formal verification, ending with constant monitoring and incident response.

Therefore, we encourage the implementation of security mechanisms at all stages of development and maintenance.

5. Vulnerabilities

[YINE-c7850b7c-H01] Invalid decimals used to calculate total assets

HIGH FIXED

Retest (2025-01-02)

The vulnerability has been fixed as recommended. The first scenario is handled because all assets are now converted to base through `_convertAssetToBase` in L706. The second scenario is handled by adding new requirements in a separate commit in L592.

Affected files

- BaseVault.sol#L612

Description

The `processAccounting` function is responsible for updating the amount of assets held in the vault. It iterates through all assets associated with the vault, retrieves their rates, and multiplies them by the vault's balance. However, there are two key issues related to decimal handling in this computation.

```
598 function processAccounting() public virtual {
599         VaultStorage storage vaultStorage = _getVaultStorage();
600
601         uint256 totalBaseBalance = vaultStorage.countNativeAsset ? address(this).
                balance : 0;
602
603         AssetStorage storage assetStorage = _getAssetStorage();
604         address[] memory assetList = assetStorage.list;
605         uint256 assetListLength = assetList.length;
606         uint256 baseAssetUnit = 10 ** (assetStorage.assets[asset()].decimals);
607
608         for (uint256 i = 0; i < assetListLength; i++) {
609             uint256 balance = IERC20(assetList[i]).balanceOf(address(this));
610             if (balance == 0) continue;
611             uint256 rate = IProvider(provider()).getRate(assetList[i]);
612             totalBaseBalance += balance.mulDiv(rate, baseAssetUnit, Math.Rounding
```

```
                .Floor);
613         }
614
615         _getVaultStorage().totalAssets = totalBaseBalance;
616         emit ProcessAccounting(block.timestamp, totalBaseBalance);
617     }
```

First (L612), the calculation `amount * rate` uses the decimal places of the base asset rather than those of the iterated asset. If the decimal configuration of the base asset differs from that of the iterated asset, it can cause the `totalAssets` value to be incorrect by several orders of magnitude. This error could result in an inflated or deflated `totalAssets`, depending on whether the base asset has more or fewer decimal places than the iterated asset.

Note: The first scenario, with different assets' decimals, was identified initially by YieldNest team at the beginning of tests.

Second (L615), the native asset is added to `totalAssets` without any adjustments for its decimal places. This can create additional inaccuracies if the base asset's decimal configuration differs from that of the native asset, again leading to potential inflation or deflation of the `totalAssets` value.

Attack scenario

An attacker could exploit these decimal-related vulnerabilities to improperly derive assets from the protocol by following these steps:

① Assume the Vault has a base asset with 6 decimal places and a native asset with 18 decimal places, with `countNativeAsset` set to true.
② Several users deposit 990,000 base assets, minting 990,000 shares.
③ The attacker deposits 10,000 base assets, acquiring 10,000 shares.
④ The `totalAssets` now amounts to 1,000,000 tokens.
⑤ The attacker sends 0.00001 (1e-5) native asset to the vault.
⑥ The attacker executes the `processAccounting` function. The protocol computes the base asset amount as 1e12 (1,000,000 * 1e6) and the native amount figures at 1e13 (since 18 - 5 = 13), combining to yield a `totalAssets` of 11,000,000 tokens (1.1e13).
⑦ The attacker redeems their 10,000 shares, which are now valued at 11 tokens each, calculated by dividing the 1.1e13 `totalAssets` by 1e12 shares.
⑧ The vault voids the attacker's shares and transfers 10,000 * 11 = 110,000 base assets. Thus, the attacker effectively steals 100,000 base assets for the cost of 0.00001 native asset.

Result of the attack: The primary outcome of this vulnerability is that the `totalAssets` variable will be inaccurately calculated, leading to theft of assets or potential losses for users due to undervalued `totalAssets`.

Recommendation

- When calculating the value of ERC20 assets, use the decimal places of the iterated asset, not the base asset's decimals, in the computation of `amount * rate`.
- Normalize the amount of the native asset to align it with the base asset's decimal places.

References

1. SCSVS I2: Token

[YINE-c7850b7c-M01] Inflating vault's share rate via reentrancy

MEDIUM FIXED

Retest (2025-01-06)

The token transfer occurs after the shares are burned in L425.
The `nonReentrant` modifier has been added to `processAccounting` function as recommended in L683.
The CEI pattern was applied for `KernelStrategy` in L419.

Affected files

- KernelStrategy.sol#L327
- BaseVault.sol#L598-L617

Description

The withdrawal process from the Max Vault works in the following way:

1. it decreases the `totalAssets` metric

```
vaultStorage.totalAssets -= _convertAssetToBase(asset_, assets);
```

2. withdraws from the buffer strategy (`KernelStrategy` vault).

```
IStrategy(vaultStorage.buffer).withdraw(assets, receiver, address(this));
```

3. burns the user's shares in the Max Vault.

```
_burn(owner, shares);
```

The withdrawal mechanism in `KernelStrategy` is similar, but it distributes the base asset to

the user (defined as the `receiver` parameter) instead of withdrawing from another vault.

```
1 SafeERC20.safeTransfer(IERC20(asset_), receiver, assets);
```

If the base asset is an ERC20 token equipped with a callback function, the user could designate a contract as the receiver. This contract can then invoke the `processAccounting` function, which updates the `totalAssets` variable in the Max Vault. Since the buffer strategy burns shares only after the transfer, the shares in the Max Vault have not yet been burned. This allows the `totalAssets` variable to revert to its pre-withdrawal value.

As a result, the user's shares in the Max Vault are subsequently burned, leading to an inflation in share value (the `totalAssets` remains constant while the number of shares decreases).

Note: This issue has been reported as medium because the currently used assets do not pose such threat, although the team plans to dynamically expand their offer.

Attack scenario

Potential attackers could follow these steps:

① Assume the Max Vault contains 1,000,000 base assets and shares, reflecting a 1:1 ratio of assets to shares. The attacker holds 100,000 shares.
② The attacker initiates a withdrawal by calling the `withdraw` function with 50,000 assets specified as the `assets` parameter and their contract as `receiver`.
③ The vault reduces `totalAssets` by 50,000, adjusting it to 950,000.
④ The vault withdraws 50,000 base assets from the buffer strategy, transferring them to the attacker.
⑤ The buffer strategy decreases its own `totalAssets`.
⑥ The buffer strategy transfers the tokens to the attacker before burning the shares of the Max Vault.
⑦ The transferred token features a callback function that triggers the attacker's contract, which calls the `processAccounting` function on the Max Vault.
⑧ The Max Vault recalculates its `totalAssets`, including shares that have yet to be burned, reverting the `totalAssets` to a value near 1,000,000.
⑨ The buffer strategy then proceeds to burn the Max Vault's shares.
⑩ The Max Vault subsequently burns the user's shares.
⑪ Consequently, the share value in the Max Vault becomes approximately 1.05, representing a 5
⑫ Further withdrawals by the attacker will yield more assets than they are entitled to.

Result of the attack: This attack results in the theft of assets alongside an inflated share value.

Recommendation

To mitigate this issue, it is essential to implement the Checks-Effects-Interactions pattern. This ensures that the transfer of tokens to the user occurs only after the shares have been fully burned.
Additionally, the `processAccounting` function should incorporate a `nonReentrant` modifier to prevent potential reentrancy attacks.

References

1. SCSVS G6: Communications

[YINE-c7850b7c-M02] Invalid amounts of asset returned by strategies

MEDIUM FIXED

Retest (2025-01-06)

The vulnerability has been fixed. Available assets are now considered in L123 and L177.

Affected files

- KernelStrategy.sol#L134-L140
- KernelStrategy.sol#L147-L147
- KernelStrategy.sol#L161-L167

Description

The `KernelStrategy` implementations allow for the withdrawal of all supported assets, as opposed to Max Vaults that can only withdraw the base asset. These strategies are designed to comply with ERC4626 by implementing its functions (such as `maxWithdraw`, `maxRedeem`, `previewWithdraw`, and `previewRedeem`). In addition, they also incorporate similar functions that handle assets beyond just the base asset, namely `maxWithdrawAsset`, `previewWithdrawAsset`, and `previewRedeemAsset`.

A key issue with these functions is their assumption that all shares in the strategy were minted for a single asset. For instance, when the `maxWithdraw` function is invoked, the contract presumes that all shares correspond to the base asset, potentially resulting in an inflated withdrawal amount as it neglects to consider that the amount needs to be apportioned among multiple assets.

Vulnerable scenario

The following sequence of actions may lead to the identified problem:

① An allocator wishes to withdraw BNBX from ynBNBk.
② The allocator calls the `maxWithdrawAsset` function for BNBX.
③ The strategy calculates the total value of all shares in terms of BNBX.
④ The allocator then invokes the `withdrawAsset` function using the amount provided.
⑤ The protocol computes the BNBX amount through the `previewWithdrawAsset` function.
⑥ The protocol checks the available balance of BNBX. If the balance is less than the requested amount, it proceeds to unstack BNBX from Kernel.
⑦ Since some shares were minted with the SLISBNB token as a deposit, there may not be sufficient BNBX available for withdrawal from Kernel, resulting in a transaction revert.

Result: This issue prevents the successful withdrawal of assets based on the values returned by the ERC4626 functions and their equivalent versions for different assets.

Recommendation

Implement a method to track the amount of deposited assets for each asset type in addition to the `totalAssets` variable, and utilize this information within the affected functions.

References

1. SCSVS C4: Vault

[YINE-c7850b7c-M03] Fee on withdrawal from strategy can lead to protocol's loss

MEDIUM ACKNOWLEDGED

Retest (2025-01-02)

Issue is acknowledged by the team with the following response:
No ynBNBx or ynETHx strategies are deployed with with fees that are greater than 0. We add strategies with no fees only.

Affected files

- KernelStrategy.sol#L175-L178
- KernelStrategy.sol#L186-L190

Description

The Max Vaults allocate assets to `KernelStrategy` deployed contracts that stake assets in Kernel vaults. When a user requests to withdraw assets from the Max Vault, those assets are taken from a special `KernelStrategy` contract called the buffer strategy (i.e. ynWBNBk). However, when the funds in buffer strategy are not sufficient, the team has to withdraw funds from other strategies (e.g. ynBNBk).

The `KernelStrategy` contracts include a fee structure for withdrawals which does not work properly for all cases. During withdrawal from `KernelStrategy` to Max Vault and to the Buffer Strategy, the fee should not be applied. This is essential to prevent an increase in the share rate in the strategy from which the assets are withdrawn (causing a situation where more shares are burned than assets withdrawn) and ensure that other users who have deposited into the strategy do not unintentionally share in the fee distribution.

Attack scenario

Attackers could execute the following sequence:

① The attacker deposits assets into the ynBNBk strategy which accepts deposits from anyone.
② The attacker subsequently deposits assets into the Max Vault.
③ The team deposits assets from Max Vault into the ynBNBk strategy.
④ The attacker initiates a withdrawal of tokens back to the Max Vault.
⑤ There are no enough assets in the buffer strategy and the team has to withdraw assets from other strategies (e.g. ynWBNBk), incurring a fee that influences the share rate.
⑥ The attacker repeats steps 2-5.
⑦ The attacker withdraws assets from the yBNBk that have a higher value than the initial deposit in step 1.

Result of the attack: The protocol experiences a loss of assets proportional to the fees charged.

Recommendation

Adjust the fee calculation functions to exclude Max Vaults, Buffer Vault, and other Yield-Nest vaults from the fee structure.

References

1. SCSVS G4: Business logic

[YINE-c7850b7c-M04] Inability to unstake required assets from Kernel

MEDIUM ACKNOWLEDGED

Retest (2025-01-02)

Issue is acknowledged by the team with the following response:
This is intended behavior. We are in touch with the kernel team and the Lista DAO team and the have confirmed the fee is not being raised soon.
Since the integration with Kernel may change in a notable way at that point, this is the best we can do now to treat this case with regards to the fee.
The way Kernel is built assumes now there are no fees - we will change this when the Kernel team notifies us.
In addition to maintaining contact and collecting updates from Kernel team, set up fee parameter monitoring to independently observe changes.

Affected files

- KernelClisStrategy.sol#L94

Description

The `KernelClisStrategy` contract is designed to stake deposited WBNB into the Kernel clisBNB vault. It happens automatically if the `syncDeposit` option is enabled. When an allocator initiates a withdrawal from the strategy, the contract verifies if the balance is adequate to fulfill the request.

If the balance is insufficient, which is often the case due to the strategy's approach of predominantly staking assets in the Kernel vault, the contract attempts to unstake the required amount from Kernel.

```
1  IStakerGateway(strategyStorage.stakerGateway).unstakeClisBNB(assets -
       vaultBalance, referralId);
```

However, the actual amount that is unstaked can differ from the requested amount. This discrepancy may arise in two scenarios:

1. If the Lista DAO team has set a fee greater than zero (it was set to zero during the audit),
2. If the available assets in Lista DAO strategies are below the requested withdrawal amount.

In such instances, the team must manually withdraw assets using the `processor` function to satisfy the withdrawal request.

Result: This leads to the inability to automatically withdraw WBNB from the `KernelClisStrategy`

Recommendation

Addressing this issue is complex. When a fee is applied in Lista DAO or when Kernel returns less WBNB than anticipated, it indicates that YieldNest has fewer assets available than expected.
To mitigate this, YieldNest could either **offset the shortfall using treasury funds** or **pass the loss on to users**.
Offset the shortfall using treasury funds: the strategy contract should verify the actual amount unstaked and draw the remaining balance from the treasury.
Pass the loss on to users: the strategy contract may either provide a lesser amount than requested (which would not comply with ERC4626) or request a larger quantity of assets from Kernel and adjust the total assets managed by the strategy accordingly. This approach would necessitate knowing the additional amount needed for the Kernel gateway request, but the strategy vault could establish a fixed additional percentage based on the Lista DAO fee, updating it as necessary.

References

1. SCSVS C4: Vault

[YINE-c7850b7c-M05] Rate provider for ynBNBx is missing ynClisBNBk strategy

MEDIUM FIXED

Retest (2025-01-02)

The vulnerability has been removed as recommended. The ynClisBNBk is now included in L18.

Affected files

- Provider.sol#L18

Description

The `Provider` contract, responsible for retrieving rates for all assets held by the vault, currently does not include the ynClisBNBk strategy. As a result, when the `processAccounting` function attempts to update the `totalAssets` variable, it fails and reverts with each call.

```
18 if (asset == MC.BUFFER || asset == MC.YNBNBk) {
19     return IERC4626(asset).previewRedeem(1e18);
20 }
```

Result: The `processAccounting` function will revert on every invocation within the `ynBNBx` context.

Recommendation

Incorporate the ynClisBNBk strategy into the list of assets supported by the `Provider` contract on the BNB Chain.

References

1. SCSVS C4: Vault

[YINE-c7850b7c-M06] Invalid rate for YieldNest vaults

MEDIUM FIXED

Retest (2025-01-02)

The vulnerability has been removed as recommended. The `convertToAssets` function is now used in L18 and L34.

Affected files

- Provider.sol#L25
- Provider.sol#L19

Description

The rate providers for Max Vaults (ynBNBx and ynETHx) have the `previewRedeem` function to determine the rate for a single strategy token. However, this function includes the withdrawal fee, which results in an incorrect valuation that is lower than the actual value of one token.

Result: The valuation of vault tokens is inaccurately represented, leading to a decreased value of the assets held within the Max Vault.

Recommendation

To ensure accurate rate evaluation, utilize the `convertToAssets` function instead.

References

1. SCSVS C4: Vault

[YINE-c7850b7c-M07] Vault not compliant with ERC4626 in maxWithdraw function

MEDIUM FIXED

Retest (2025-01-06)

The vulnerability has been fixed as recommended in BaseVault. The `maxWithdraw` function now includes fees in the calculations via the `previewRedeem` function in L157. The issue was also removed from KernelStrategy via `previewRedeemAsset` in L121.

Affected files

- KernelStrategy.sol#L134-L140
- BaseVault.sol#L143-L157

Description

Per the ERC4626 standard, the `maxWithdraw` function is required to return the maximum amount of assets that can be withdrawn without causing a revert, and this amount must not exceed the actual maximum that can be accepted.

In the `BaseVault` and `KernelStrategy` contracts, the `maxWithdraw` function has not been updated to include fees. As a result, it calculates the total assets without factoring in the fees, which may cause the calculated shares to exceed the user's available shares when they attempt to withdraw the maximum amount suggested by `maxWithdraw`.

Vulnerable scenario

The following actions illustrate this issue:

① An external application integrates with Max Vault, adding a feature for users to withdraw their total assets.
② A user initiates a request to withdraw all their assets.

③ The external application calls `maxWithdraw` to determine the total withdrawable assets for that user, then invokes the `withdraw` function.
④ Max Vault computes the shares to be burnt using the `previewWithdraw` function, which results in a calculation that suggests burning more shares than the user possesses.
⑤ The transaction reverts during the `_burn` function call.

Result: This results in non-compliance with the ERC4626 standard, leading to the potential for withdrawal attempts based on `maxWithdraw` results to fail.

Recommendation

The `maxWithdraw` function should be updated to account for fees, thus providing a more accurate amount of assets that a user may withdraw.

References

1. ERC-4626: Tokenized Vaults
2. SCSVS G4: Business logic

[YINE-c7850b7c-M08] Back-running the rewards transfer

MEDIUM FIXED

Retest (2025-01-03)

The team has implemented the `alwaysComputeTotalAssets` flag in L628 to recalculate TVL on every change in environments where gas cost is not an issue and also plans to use protected RPC as recommended.

Affected files

- BaseVault.sol#L598-L617

Description

An attacker can unfairly gain by back-running the transfer with rewards that have not yet been included in `totalAssets`. This appears due to the assumption that rewards are accumulated discretely.

The `processAccounting` function is designed to update total assets by iterating through the asset list to retrieve and calculate their balances and rates.

```
598 function processAccounting() public virtual {
599     VaultStorage storage vaultStorage = _getVaultStorage();
```

```
600
601     uint256 totalBaseBalance = vaultStorage.countNativeAsset ? address(this).
            balance : 0;
602
603     AssetStorage storage assetStorage = _getAssetStorage();
604     address[] memory assetList = assetStorage.list;
605     uint256 assetListLength = assetList.length;
606     uint256 baseAssetUnit = 10 ** (assetStorage.assets[asset()].decimals);
607
608     for (uint256 i = 0; i < assetListLength; i++) {
609         uint256 balance = IERC20(assetList[i]).balanceOf(address(this));
610         if (balance == 0) continue;
611         uint256 rate = IProvider(provider()).getRate(assetList[i]);
612         totalBaseBalance += balance.mulDiv(rate, baseAssetUnit, Math.Rounding.
                Floor);
613     }
614
615     _getVaultStorage().totalAssets = totalBaseBalance;
616     emit ProcessAccounting(block.timestamp, totalBaseBalance);
617 }
```

However, the timing of the reward distribution can create an opportunity for an attacker to benefit unfairly. While the function's purpose is to maintain correct accounting after rewards are distributed, its current design allows an arbitrage opportunity.

After the reward transfer, an attacker can make a significant deposit before the `processAccounting` function updates the asset rate, then withdraw immediately. Thereby unfairly sharing in the rewards generated by other users' contributions or withdrawal fees.

In order to make this attack profitable, the increase in share price must be greater than the fee paid for such transaction. The price increase depends on the size of rewards being distributed at once, the vault's TVL and the size of attacker's deposit.

Note: This issue was also identified internally by YieldNest team independently, during the security review.

Vulnerable scenario

The following actions illustrate this issue:

① There are 100 tokens deposited, with a rate of 1 share for every 2 tokens (1:2).
② Then, 10 tokens are transferred as rewards.
③ Before the `processAccounting` function is executed, the attacker deposits another 100 tokens, thus receiving 50 shares.
④ Once `processAccounting` is called, the rate adjusts to 10:21.

⑤ The attacker then withdraws 105 tokens for their 50 shares, effectively acquiring 50% of the rewards.

The similar arbitrage opportunity can appear when over 50% of Vault's TVL is being withdrawn within a short period of time. The root cause is that the fee (assets) is help in the vault and increases the share price by more than the fee value.

Result: The attacker acquires rewards that are generated by other users' capital.

Recommendation

- Control the size of rewards being added to the vault and make sure it does not make the share's price increase by more than the fee.
- Schedule `processAccounting` to be called at least daily (or as often as possible) to facilitate smooth reward distribution and to reduce the potential for arbitrage.
- Implement the transfer of rewards through protected RPC (MEV resistant). This would ensure that the attacker's cannot see the transaction that transfers rewards and detect the opportunity. However, if the rewards distribution depend on other publicly visible events, the attacker would still be able to detect the opportunity.
- Additionally, consider whether `processAccounting` should only be available to a role that is trusted, as this will prevent an attacker from using flashloans to create a leverage.

References

1. SCSVS G1: Architecture, design and threat modeling
2. SCSVS G4: Business logic

[YINE-c7850b7c-L01] Ability to withdraw non-active assets

LOW FIXED

Retest (2025-01-02)

The vulnerability has been removed as recommended. The function requires now assets to be active in L275-L277.

Affected files

- KernelStrategy.sol#L305-L332

Description

Some of `KernelStrategy` contracts will represent strategies where anyone can take part - the ones with no allocators. However, users are allowed to deposit only selected assets, marked as active. On the contrary, not active assets can be withdrawn by the users. This allows users to make a free swaps and force the protocol to unstake any supported assets if the `syncWithdraw` is set to true.

Result: Anyone can make free swaps from active to non-active assets in vault and force the vault to unstake any supported asset.

Recommendation

Consider blocking withdrawals of non-active assets.

References

1. SCSVS G5: Access control

[YINE-c7850b7c-L02] Decimals should be read from asset

LOW ACKNOWLEDGED

Retest (2025-01-02)

Issue is acknowledged by the team with the following response: *This is intended behavior. This is done on purpose to accomodate adding the Kernel Vaults or other assets that don't have decimals() function as assets.*

Affected files

- ynETHxVault.sol#L14
- Vault.sol#L50
- KernelStrategy.sol#L396-L402

Description

The existing implementation of the `Vault` assigns the vaults's decimals during the initialization based on the parameter. Additionally, `KernelStrategy` contract adds Kernel vaults using `addAssetWithDecimals` which also accepts the decimals as a parameter.

This approach can lead to a situation when the wrong decimals are passed by mistake in the parameter are stored in the contract. Later, the vault would calculate incorrect values for rates. Morever, when external application integrate with vaults, they would use wrong

decimals as vault's decimals.

Result: Decimal values that do not accurately reflect their actual representations.

Recommendation

- To enhance adaptability and mitigate the likelihood of invalid values, set the vault's decimals when the first asset is added, or alternatively, retrieve this information directly on demand from the asset itself.
- When adding new asset (e.g. Kernel vaults), retrieve it's decimals from the vault.

References

1. SCSVS G1: Architecture, design and threat modeling

[YINE-c7850b7c-L03] Lack of clear limitations on fees

LOW ACKNOWLEDGED

Retest (2025-01-02)

Issue is acknowledged by the team with the following response: *This is intended behavior. We left it as such for flexibility.*

Affected files

- Vault.sol#L53
- Vault.sol#L89
- ynETHxVault.sol#L18
- MigratedKernelStrategy.sol#L87-L87

Description

The fee imposed on withdrawals can be set without any limits on its maximum value.

Result: Setting a very high fee may consume user funds or lead to DoS of the smart contract.

Recommendation

Set a predetermined maximum value for `baseWithdrawalFee`.

References

1. SCSVS G4: Business logic

[YINE-c7850b7c-L04] Inability to deposit and withdraw different asset than the base

LOW ACKNOWLEDGED

Retest (2025-01-02)

Issue is acknowledged by the team with the following response: *The strategy is meant to be used only with WBNB staked as clis. It will not receive other assets.*

Affected files

- KernelClisStrategy.sol#L91
- KernelClisStrategy.sol#L50

Description

The `KernelClisStrategy` contract is designed to accommodate WBNB, which is staked in Kernel as clisBNB tokens. However, due to its inheritance from `Vault`, the contract has the potential to handle multiple asset types for deposits and withdrawals. The introduction of additional asset support can lead to transactions failures based on the settings of `syncDeposit` and `syncWithdraw`.

When `syncDeposit` is activated, only WBNB can be deposited. If another token is attempted, the contract will execute a withdrawal of BNB in WBNB, which could lead to revert. Conversely, with `syncWithdraw` enabled, if the vault's balance is low, trying to withdraw assets other than WBNB will result in a transaction failure.

The `processor` function's capability to transfer locked tokens adds a layer of risk.

Note: The severity of this issue has been reduced because the inclusion of additional assets is not currently anticipated.

Result: Transactions failures on deposits and withdrawals.

Recommendation

Since the `KernelClisStrategy` is intended solely for WBNB, the `addAsset` function should be overridden to exclusively allow WBNB as active asset and the Kernel clisBNB vault as a non-active asset.

References

1. SCSVS G4: Business logic

[YINE-c7850b7c-L05] Invalid modifier in KernelClisStrategy

LOW FIXED

Retest (2025-01-02)

The vulnerability has been removed. The `KernelClisStrategy` no longer overrides `_deposit` and `_withdrawAsset` functions. It has been replaced with minimal implementation.

Affected files

- KernelClisStrategy.sol#L35
- KernelClisStrategy.sol#L80

Description

The `_deposit` and `_withdrawAsset` functions in the `KernelClisStrategy` contain an access control modifier that is not suitable for the intended functionality. Although the current implementation restricts access to only users with the `ALLOCATOR_ROLE`, it fails to account for the changes made by the `setHasAllocator` function.

Vulnerable scenario

The following steps outline the vulnerability:

① The allocator manager invokes `setHasAllocator` and sets the value to false.
② Despite this setting, the functions can still be called exclusively by users with the `ALLOCATOR_ROLE`, rendering the modification irrelevant.

Result: The management of the strategy is still limited to users with the `ALLOCATOR_ROLE`, irrespective of the state set by the `setHasAllocator` function.

Recommendation

Replace the `onlyRole` modifier with a more appropriate `onlyAllocator` modifier.

References

1. SCSVS G5: Access control

6. Recommendations

[YINE-c7850b7c-R01] Remove hardcoded rate for SolvBTC.BBN

INFO IMPLEMENTED

Retest (2025-01-02)

The recommendation has been implemented as recommended in L39.

Description

The `BTCRateProvider` rate provider uses hardcoded `1e18` rate for SolvBTC.BBN token. The price of SolvBTC.BBN is pegged 1:1 to BTC, but it supports retrieval of its current value on-chain and therefore it is a good practice to follow it, the same way as it's done for frxETH.

Recommendation

Use `getValueByShares` function to retrieve the value of 1 SolvBTC.BBN token.

References

1. SCSVS G1: Architecture, design and threat modeling

[YINE-c7850b7c-R02] Remove unnecessary code

INFO IMPLEMENTED

Retest (2025-01-02)

The recommendation has been implemented.

Description

There are the following elements in the code that should not be present in the production-ready code:

- unused functions,
- unnecessary function calls,
- TODOs.

Recommendation

- Remove the call to `safeIncreaseAllowance` (KernelClisStrategy.sol#L51), because the asset is later set as `msg.value`.
- Remove the unused function `_bufferMaxSize` (Vault.sol#L78).

The TODOs in the code are related to values which are not known yet and some of them will be known during the deployment process. It is important to address them at that time.

References

1. SCSVS G1: Architecture, design and threat modeling

[YINE-c7850b7c-R03] Use getPrices instead of getCurveEmaEthPerFrxEth

INFO ACKNOWLEDGED

Retest (2025-01-02)

The recommendation has been acknowledged by the team.

Description

The existing implementation of the `Provider` retrieves the price for the `MC.SFRXETH` asset via the `getCurveEmaEthPerFrxEth` function. This method is inherently reliant on parameters that might change over time, specifically the fee and asset ratios.

Consequently, this could lead to potential price manipulation stemming from discrepancies in the `last_price` calculation. In situations where the price diverges significantly from a 1:1 ratio or when fees are elevated, the `last_price` may be inaccurately computed.

At present, the fee is set at `2000000` (0.02%), and the ratio is approximately 1:1, which mitigates immediate risk.

Recommendation

Transition from utilizing the `getCurveEmaEthPerFrxEth` function to the `getPrices` function. The latter not only retrieves price information but also incorporates validation checks to ensure the accuracy of the returned price.

References

1. Daniel Von Fange - Curve has the risk of oracle manipulation

2. SCSVS G1: Architecture, design and threat modeling

[YINE-c7850b7c-R04] Emit events for important state changes

INFO IMPLEMENTED

Retest (2025-01-02)

The recommendation has been implemented as recommended in L104.

Description

It is important to ensure that updates to critical parameters are logged through events for better tracking and transparency.

Recommendation

Implement an event emission in the following function to capture updates:

- setBaseWithdrawalFee

References

1. SCSVS G1: Architecture, design and threat modeling
2. Principles and Best Practices to Design Solidity Events in Ethereum and EVM

7. Impact on risk classification

Risk classification is based on the one developed by OWASP [1], however it has been adapted to the immutable and transparent code nature of smart contracts. The Web3 ecosystem forgives much less mistakes than in the case of traditional applications, the servers of which can be covered by many layers of security.

Therefore, the classification is more strict and indicates higher priorities for paying attention to security.

OVERALL RISK SEVERITY				
Impact on risk	HIGH	CRITICAL	HIGH	MEDIUM
	MEDIUM	MEDIUM	MEDIUM	LOW
	LOW	LOW	LOW	INFO
		HIGH	MEDIUM	LOW
		Likelihood		

[1] OWASP Risk Rating methodology

8. Long-term best practices

8.1. Use automated tools to scan your code regularly

It's a good idea to incorporate automated tools (e.g. slither) into the code writing process. This will allow basic security issues to be detected and addressed at a very early stage.

8.2. Perform threat modeling

Before implementing or introducing changes to smart contracts, perform threat modeling and think with your team about what can go wrong. Set potential targets of the attacker and possible ways to achieve them, keep it in mind during implementation to prevent bad design decisions.

8.3. Use Smart Contract Security Verification Standard

Use proven standards to maintain a high level of security for your contracts. Treat individual categories as checklists to verify the security of individual components. Expand your unit tests with selected checks from the list to be sure when introducing changes that they did not affect the security of the project.

8.4. Discuss audit reports and learn from them

The best guarantee of security is the constant development of team knowledge. To use the audit as effectively as possible, make sure that everyone in the team understands the mistakes made. Consider whether the detected vulnerabilities may exist in other places, audits always have a limited time and the developers know the code best.

8.5. Monitor your and similar contracts

Use the tools available on the market to monitor key contracts (e.g. the ones where user's tokens are kept). If you have used code from another project, monitor their contracts as well and introduce procedures to capture information about detected vulnerabilities in their code.

Damian Rusinek

Smart Contracts Auditor

@drdr_zz

damian.rusinek@composable-security.com

Paweł Kuryłowicz

Smart Contracts Auditor

@wh01s7

pawel.kurylowicz@composable-security.com