hexens × TOKEMAK

MAR.25

SECURITY REVIEW REPORT FOR TOKEMAK

CONTENTS

ABOUT HEXENS

Hexens is a cybersecurity company that strives to elevate the standards of security in Web 3.0, create a safer environment for users, and ensure mass Web 3.0 adoption.

Hexens has multiple top-notch auditing teams specialized in different fields of information security, showing extreme performance in the most challenging and technically complex tasks, including but not limited to: Infrastructure Audits, Zero Knowledge Proofs / Novel Cryptography, DeFi and NFTs. Hexens not only uses widely known methodologies and flows, but focuses on discovering and introducing new ones on a day-to-day basis.

In 2022, our team announced the closure of a $4.2 million seed round led by IOSG Ventures, the leading Web 3.0 venture capital. Other investors include Delta Blockchain Fund, Chapter One, Hash Capital, ImToken Ventures, Tenzor Capital, and angels from Polygon and other blockchain projects.

Since Hexens was founded in 2021, it has had an impressive track record and recognition in the industry: Mudit Gupta - CISO of Polygon Technology - the biggest EVM Ecosystem, joined the company advisory board after completing just a single cooperation iteration. Polygon Technology, 1inch, Lido, Hats Finance, Quickswap, Layerswap, 4K, RociFi, as well as dozens of DeFi protocols and bridges, have already become our customers and taken proactive measures towards protecting their assets.

EXECUTIVE SUMMARY

OVERVIEW

This report covered updates to the smart contracts of Tokemak V2. It introduced various new functionality, such as new destination vaults and support for base assets with less than 18 decimals.

Our security assessment was a full review of the smart contracts spanning a total of 3 weeks.

During our audit, we have identified two medium severity vulnerabilities, which could have resulted in minor impact in some edge cases.

We have also identified several minor severity vulnerabilities and code optimisations.

All of our reported issues were fixed by the development team and consequently validated by us.

We can confidently say that the overall security and code quality have increased after completion of our audit.

SCOPE

The analyzed resources are located on:

https://github.com/Tokemak/v2-core/tree/9e3ce2fdf762a80b3fdb042c5741a4939266ab3b

The issues described in this report were fixed in the following commits:

https://github.com/Tokemak/v2-core/tree/c53436c9758231347109b6c39f18778eb94effe3

AUDITING DETAILS

STARTED
03.03.2025

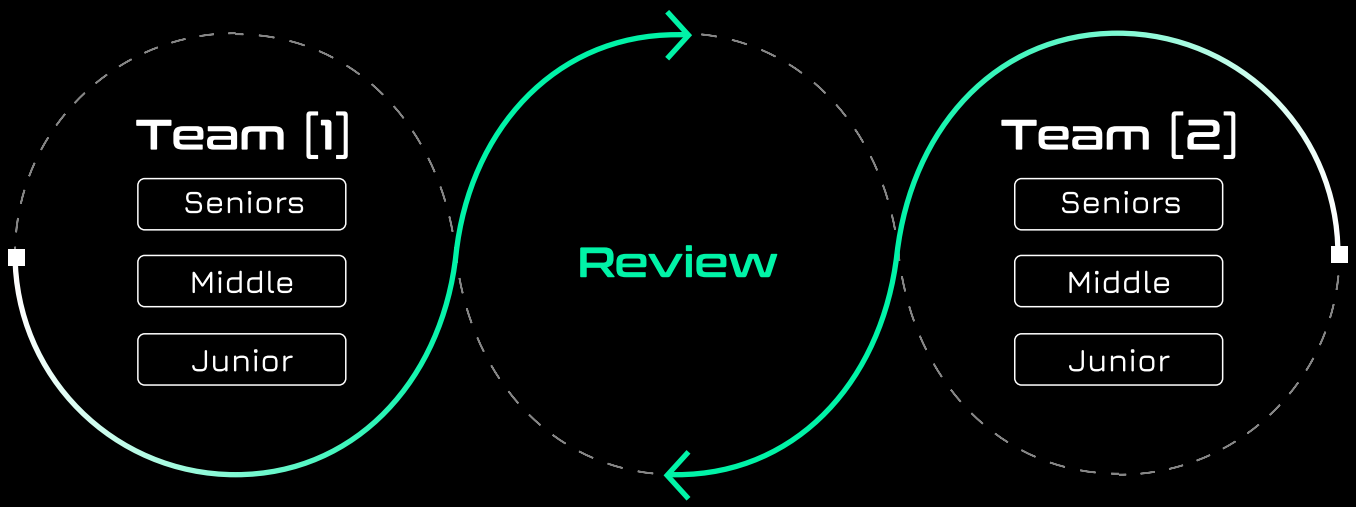
DELIVERED
25.03.2025

Review Led by

KASPER ZWIJSEN
Head of Audits | Hexens

HEXENS METHODOLOGY

Hexens methodology involves 2 teams, including multiple auditors of different seniority, with at least 5 security engineers. This unique cross-checking mechanism helps us provide the best quality in the market.



SEVERITY STRUCTURE

The vulnerability severity is calculated based on two components

- Impact of the vulnerability
- Probability of the vulnerability

Impact	Probability			
	rare	unlikely	likely	very likely
Low/Info	Low/Info	Low/Info	Medium	Medium
Medium	Low/Info	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

SEVERITY CHARACTERISTICS

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities with this level of severity can result in significant financial losses or reputational damage. They often allow an attacker to gain complete control of a contract, directly steal or freeze funds from the contract or users, or permanently block the functionality of a protocol. Examples include infinite mints and governance manipulation.

High

Vulnerabilities with this level of severity can result in some financial losses or reputational damage. They often allow an attacker to directly steal yield from the contract or users, or temporarily freeze funds. Examples include inadequate access control integer overflow/underflow, or logic bugs.

Medium

Vulnerabilities with this level of severity can result in some damage to the protocol or users, without profit for the attacker. They often allow an attacker to exploit a contract to cause harm, but the impact may be limited, such as temporarily blocking the functionality of the protocol. Examples include uninitialized storage pointers and failure to check external calls.

Low

Vulnerabilities with this level of severity may not result in financial losses or significant harm. They may, however, impact the usability or reliability of a contract. Examples include slippage and front-running, or minor logic bugs.

Informational

Vulnerabilities with this level of severity are regarding gas optimizations and code style. They often involve issues with documentation, incorrect usage of EIP standards, best practices for saving gas, or the overall design of a contract. Examples include not conforming to ERC20, or disagreement between documentation and code.

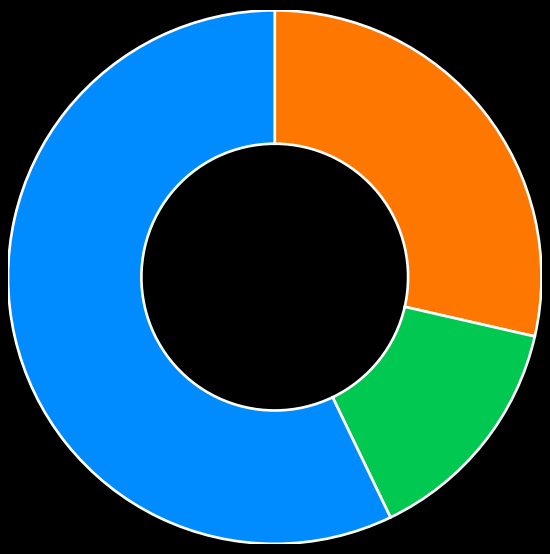
ISSUE SYMBOLIC CODES

Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

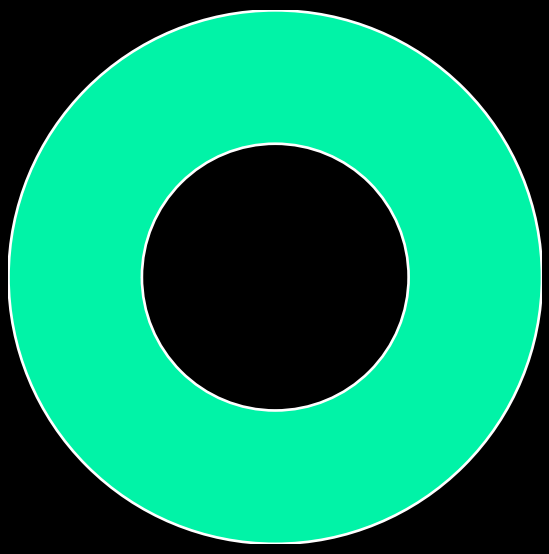
FINDINGS SUMMARY

Severity	Number of Findings
Critical	0
High	0
Medium	2
Low	1
Informational	4

Total: 7



- Medium
- Low
- Informational



- Fixed

WEAKNESSES

This section contains the list of discovered weaknesses.

TOKES-2

NO RESTRICTIONS ON INITIALIZE FUNCTION

SEVERITY: Medium

PATH:

src/stats/calculators/base/
BalancerV3StablePoolCalculatorBase.sol:initialize

REMEDIATION:

Like other functions, it should have the initializer modifier to prevent the subsequent calling of the initialize function.

STATUS: Fixed

DESCRIPTION:

There are no restrictions on calling the initialize function, so can call it at any time to modify important variables. This allows for limited changes to the poolAddress, which may result in price updates not functioning correctly.

```
/// @inheritdoc IStatsCalculator
/// @dev On this calculator, if submitting a boosted token aprId will be for
ERC4626.asset() token
function initialize(bytes32[] memory dependentAprIds, bytes memory initData)
external virtual override {
    [..]
}
```

NAVTRACKING STATE LENGTH IS INCORRECTLY INCREASED

SEVERITY: Medium

PATH:

src/strategy/NavTracking.sol:insert

REMEDIATION:

The self.len increment on lines 33-35 should be moved inside of the earlier conditional when the index is incremented.

STATUS: Fixed

DESCRIPTION:

The NavTracking library implements a State struct to hold at most 91 values of the NAV history. It also keeps an index, length and timestamp.

In the **insert** function, there is the logic that if the current timestamp is less than 1 day after then last timestamp, then it won't move the index further and overwrite the same value each time. However, the length (**len**) is still increased in this case. As a result, the **len** could be as high as 91, while the **history** array only contains an element at index 0.

This could lead to DoS reverts when the function **state.getDaysAgo** is used, as it would revert with an array out-of-bounds error.

The function is used inside of **AutopoolETHStrategy:navUpdate**, which is used by the Autopool itself. These function check the **state.len** to make sure the day entry exists, but in this case it would still lead to a revert and potential DoS.

```
    function insert(State storage self, uint256 navPerShare, uint40
timestamp) internal {
        if (timestamp < self.lastFinalizedTimestamp) revert
InvalidNavTimestamp(self.lastFinalizedTimestamp, timestamp);

        // if it's been a day since the last finalized value, then finalize
the current value
        // otherwise continue to overwrite the currentIndex
        if (timestamp - self.lastFinalizedTimestamp >= 1 days) {
            if (self.lastFinalizedTimestamp > 0) {
                self.currentIndex = (self.currentIndex + 1) %
MAX_NAV_TRACKING;
            }
            self.lastFinalizedTimestamp = timestamp;
        }

        self.history[self.currentIndex] = navPerShare;
        if (self.len < MAX_NAV_TRACKING) {
            self.len += 1;
        }

        emit NavHistoryInsert(navPerShare, timestamp);
    }
```

INCORRECT PRICING IN SLIPPAGEBUDGETHOOK COULD LEAD TO EXTRA SLIPPAGE COUNTING

SEVERITY: Low

PATH:

src/strategy/hooks/SlippageBudgetHook.sol:onRebalanceStart

REMEDIATION:

The function _getBaseValue should use the average between the safe and spot price to get the same asset price that was used for the totalAssets(Global) AUM calculation.

STATUS: Fixed

DESCRIPTION:

The SlippageBudgetHook is a hook that can set a slippage budget for an Autopool on a rebalance. For each rebalance, it can use some amount of the budget during a time period, to protect against heavy slippage.

It calculates the slippage by taking the parameter values (**assetOut**, **amountOut**) and (**assetIn**, **amountIn**) and calculates the **valueOut** and valueIn using the **_getBaseValue** function. This function using the spot price of the DestinationVault directly.

Afterwards, the difference of **valueOut** and **valueIn** is compared to the AUM, which is retrieved using **IAutopool(msg.sender).totalAssets(IAutopool.TotalAssetPurpose.Global)**. The function **totalAsset** will return the normal price for **Global**, which is calculated from the average between the safe and spot price as can be seen in **AutopoolDebt.sol:_recalculateDestInfo**:

```
(uint256 spotPrice, uint256 safePrice, bool isSpotSafe) =
destVault.getRangePricesLP();
uint256 minPrice = spotPrice > safePrice ? safePrice : spotPrice;
uint256 maxPrice = spotPrice > safePrice ? spotPrice : safePrice;
[..]
uint256 newDebtValue = (minPrice * currentShares + maxPrice * currentShares)
/ (div * 2);
```

This means that the value and the AUM are using different prices of the assets by default, depending on how much the safe and spot price diverged.

The difference will lead to some amount of default slippage every time, even though this slippage never happened. Depending on the amount, it can eat away at the slippage budget.

```
function onRebalanceStart(ProcessRebalanceParams calldata args, address)
external override {
    [..]
    // Calculate slippage in basis points
    (uint256 valueOut,) = _getBaseValue(params.destinationOut,
params.tokenOut, params.amountOut);
    (uint256 valueIn,) = _getBaseValue(params.destinationIn, params.tokenIn,
params.amountIn);
    uint256 aum =
IAutopool(msg.sender).totalAssets(IAutopool.TotalAssetPurpose.Global);

    // Slippage = (valueOut - valueIn) * 10000 / totalAssets
    uint256 slippageBps = ((valueOut - valueIn) * BPS_DENOMINATOR) / aum;
    [..]
}
```

LSTPRICEHOOK TOLERANCE IS NOT CAPPED TO A MAXIMUM

SEVERITY: Informational

PATH:

src/strategy/hooks/LstPriceHook.sol:_configureAutopool

REMEDIATION:

See description.

STATUS: Fixed

DESCRIPTION:

The LstPriceHook allows for setting a tolerance factor per Autopool to check the difference of the safe and spot price of an LST against a tolerance.

The function **_configureAutopool** sets this tolerance but does not validate the input value. Since the tolerance is a factor in **1e18** decimals and used compare the ratio between safe/spot, it should be capped to a sensible value, such as **1e18** (100%).

```
    function _configureAutopool(address autopool, uint256 tolerance) private
{
        Errors.verifyNotZero(tolerance, "tolerance");
        autopoolTolerance[autopool] = tolerance;
        emit AutopoolToleranceConfigured(autopool, tolerance);
    }
```

USAGE OF LITERALS INSTEAD OF NAMED CONSTANTS

SEVERITY: Informational

PATH:

src/strategy/hooks/LstPriceHooks.sol:onRebalanceStart

src/strategy/hooks/MinTimeGapHook.sol:_configurateAutopool

REMEDIATION:

We recommend to use named constants instead of literals to improve readability and code quality as refactoring a parameter value would be easier if it exists in multiple places.

STATUS: Fixed

DESCRIPTION:

In various function there are literals used instead of named constants:

- **onRebalanceStart** on line 108: **1e18** and **10_000**
- **_configurateAutopool** on line 154 and 157: **1 minutes** and **30 days**

```
function _configureAutopool(address autopool, uint40 minSecondsGaps, uint40
idleMinSecondsGaps) private {
    if (minSecondsGaps < 1 minutes || minSecondsGaps > 30 days) {
        revert Errors.InvalidParam("minSecondsGap");
    }
    if (idleMinSecondsGaps < 1 minutes || idleMinSecondsGaps > 30 days) {
        revert Errors.InvalidParam("idleMinSecondsGap");
    }
    [..]
}
```

SIMPLIFICATION OF HOOK UNREGISTRATION USING DELETE

SEVERITY: Informational

PATH:

MinTimeGapHook.sol:_onUnregistered

NavLookbackHook.sol:_onUnregistered

LstPriceHook.sol:_onUnregistered

REMEDIATION:

See description.

STATUS: Fixed

DESCRIPTION:

In the **_onUnregistered** function, the entry of the given Autopool is erased by writing a new struct with only zero values. We recommend to simplify this by using **delete** on the entry directly.

```
    function _onUnregistered(
        bytes memory
    ) internal override {
        autopoolData[msg.sender] = AutopoolData({
            minSecondsGap: 0,
            idleMinSecondsGap: 0,
            lastRebalanceTimestamp: 0,
            idleLastRebalanceTimestamp: 0
        });
        emit AutopoolMinTimeGapConfigured(msg.sender, 0, 0);
    }
```

NO ACCESS CONTROL IN WITHDRAWBASEASSET FUNCTION

SEVERITY: Informational

PATH:

src/vault/DestinationVault:withdrawBaseAsset#L318-L323

REMEDIATION:

See description.

STATUS: Fixed

DESCRIPTION:

The function **withdrawBaseAsset** does not perform any access control checks on the caller.

The internal function **_withdrawBaseAsset** ensures this by burning shares from the caller, but since the Autopool is the only one that can mint these shares, it could be safer to also put the same access control on **withdrawBaseAsset** in case the Autopool has an arbitrary transfer vulnerability, making the DestinationVault shares useless if they were to end up in an attacker's possession.

```
    /// @inheritdoc IDestinationVault
    function withdrawBaseAsset(
        uint256 shares,
        address to
    ) external returns (uint256 amount, address[] memory tokens, uint256[] memory tokenAmounts) {
        return _withdrawBaseAsset(msg.sender, shares, to);
    }
```


hexens
TOKEMAK