

hexens x Spool

APR.24

**SECURITY REVIEW  
REPORT FOR  
SPOOL**

# CONTENTS

- About Hexens
- Executive summary
  - Overview
  - Scope
- Auditing details
- Severity structure
  - Severity characteristics
  - Issue symbolic codes
- Findings summary
- Weaknesses
  - isRedeemEmpty should use getMaxRedeem
  - \_compound can call \_getProtocolRewardsInternal
  - Unused Import

# 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

The audit focused on the Metamorpho strategy smart contract for the Spool Protocol V2. The Spool Protocol V2 acts as a DeFi middleware enabling users to participate in a specific set of yield-generating strategies.

Our security assessment involved a comprehensive review of the strategy smart contract over a total of 1 week. Throughout this audit, we identified three minor issues.

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

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

# SCOPE

The analyzed resources are located on:

<https://github.com/SpoolFi/spool-v2-core/blob/metamorpho/src:strategies/ERC4626StrategyBase.sol>

<https://github.com/SpoolFi/spool-v2-core/blob/metamorpho/src:strategies/MetamorphoStrategy.sol>

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

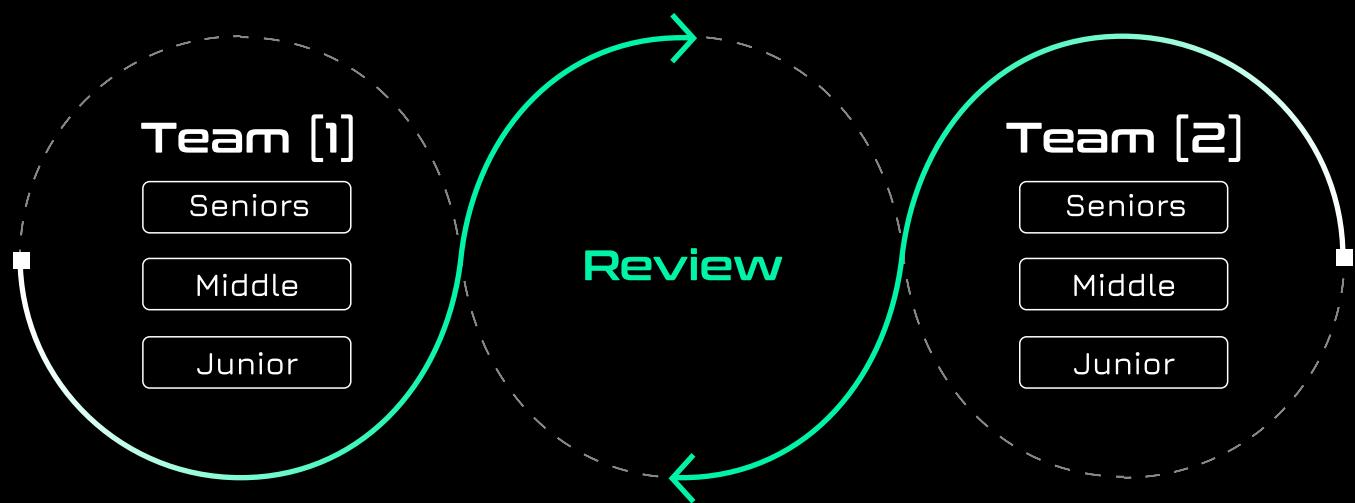
<https://github.com/SpoolFi/spool-v2-core/pull/23/commits/62b511065005e878a3e71dc8e40f0e0f6d6246fc>

# AUDITING DETAILS

	<b>STARTED</b> 29.04.2024	<b>DELIVERED</b> 06.05.2024
Review Led by	<b>MIKHAIL EGOROV</b> Senior Security Researcher   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.

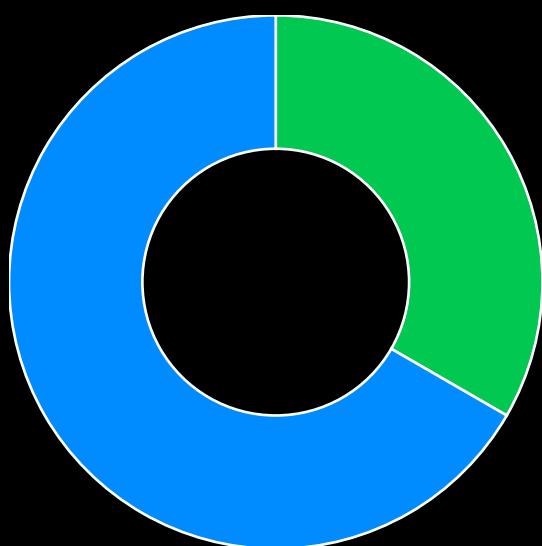
# ISSUE SYMBOLIC CODES

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.

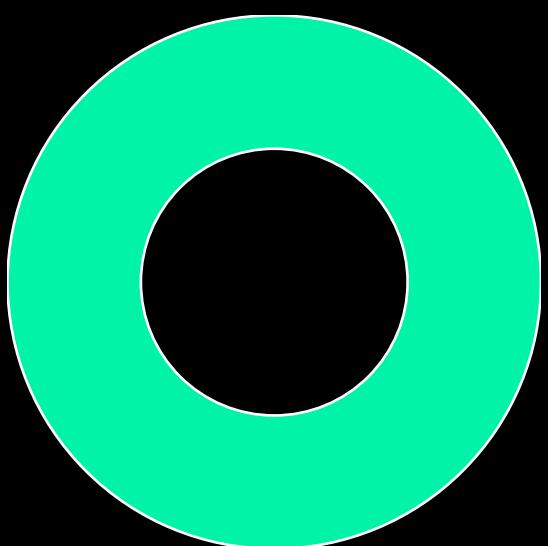
# FINDINGS SUMMARY

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

Total: 3



- Low
- Informational



- Fixed

# WEAKNESSES

This section contains the list of discovered weaknesses.

SPOOLS-3

## **ISREDEEMALEMPTY SHOULD USE GETMAXREDEEM**

SEVERITY:

Low

PATH:

src/libraries/ERC4626Lib.sol:L44

REMEDIATION:

Use ERC4626Lib.getMaxRedeem().

STATUS:

Fixed

DESCRIPTION:

The `isRedeemEmpty()` function should be updated to utilize the `ERC4626Lib.getMaxRedeem()` function instead of directly calling `vault.maxRedeem()`. Otherwise, this may result in `beforeRedeemalCheck()` failure when attempting to redeem the entire strategy balance.

SPOOLS-1

## **\_COMPOUND CAN CALL \_GETPROTOCOLREWARDSINTERNAL**

SEVERITY: Informational

PATH:

src/strategies/MetamorphoStrategy.sol:L93-L103

REMEDIATION:

See description.

STATUS: Fixed

DESCRIPTION:

The `_compound()` function can call the `_getProtocolRewardsInternal()` internal function instead of duplicating its functionality in the body. This approach will help reduce the size of the contract and streamline the code.

Change the `_getProtocolRewardsInternal()` function in the following way to transfer the reward to the `swapper`:

```
function _getProtocolRewardsInternal() internal virtual override returns
(address[] memory, uint256[] memory) {
    MetamorphoStrategyStorage memory $ = _getMetamorphoStrategyStorage();
    uint256[] memory amounts = new uint256[]($.rewards.length);
    address underlyingAsset = assets()[0];
    for (uint256 i; i < $.rewards.length; i++) {
        // for reward like underlying asset we will return 0
        // it should be calculated separately in DHW
        if (underlyingAsset != $.rewards[i]) {
            amounts[i] = IERC20($.rewards[i]).balanceOf(address(this));
            if (amounts[i] > 0) {
                IERC20($.rewards[i]).safeTransfer(address(swapper), amounts[i]);
            }
        }
    }
    return ($.rewards, amounts);
}
```

Call `_getProtocolRewardsInternal()` from `_compound()`.

## UNUSED IMPORT

SEVERITY: Informational

PATH:

src/strategies/MetamorphoStrategy.sol:L5

REMEDIATION:

Remove the unused import statement.

STATUS: Fixed

DESCRIPTION:

The **MetamorphoStrategy** imports **Roles.sol** but does not utilize it in the code.

hexens x Spool