FEB.24

SECURITY REVIEW REPORT FOR SPOOL

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

The audit conducted focused on the new Gearbox V3 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. In the Gearbox strategy, WETH or USDC is deposited into its equivalent Gearbox V3 pool for dTokens. The dTokens are then deposited into a Gearbox farming pool to receive extra rewards in the form of GEAR.

Our security assessment involved a comprehensive review of the strategy smart contract, spanning a total of 1 week. During this audit, we identified one low-severity and two informational issues. All of our reported issues were fixed or acknowledged 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/pull/17/commits/d4eec6550c60b7dc1c64f208af79469b2c4c26bf

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

https://github.com/SpoolFi/spool-v2-core/commit/8fc336ce65c044d62c56069ba0ee83ccc7d2a092

AUDITING DETAILS

STARTED

19.02.2024

DELIVERED

23.02.2024

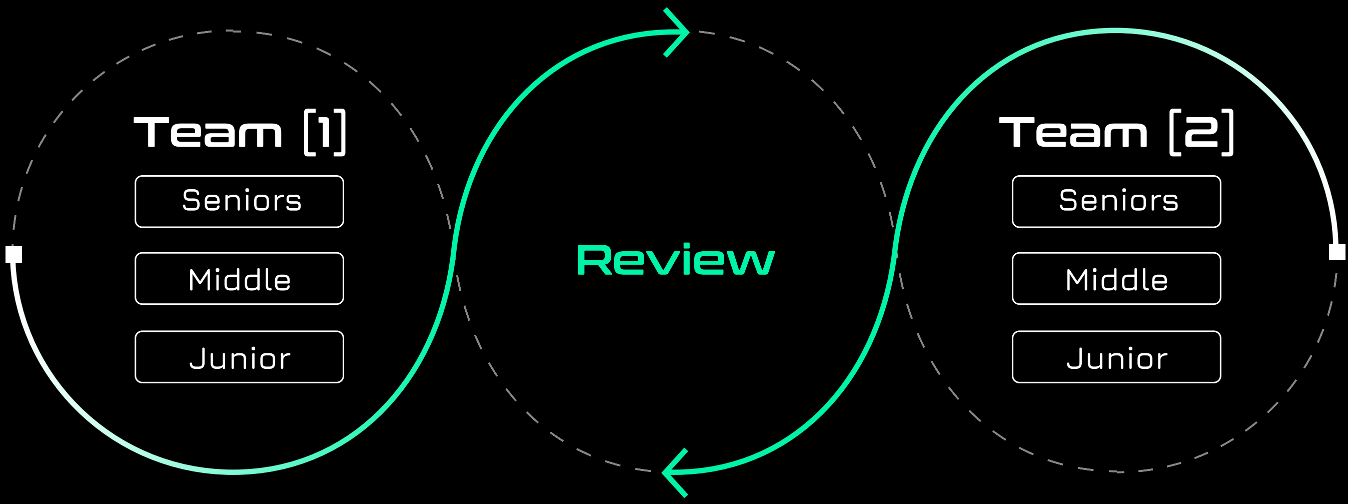
Review Led by

MIKHAIL EGOROV

Lead Smart Contract Auditor | 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.

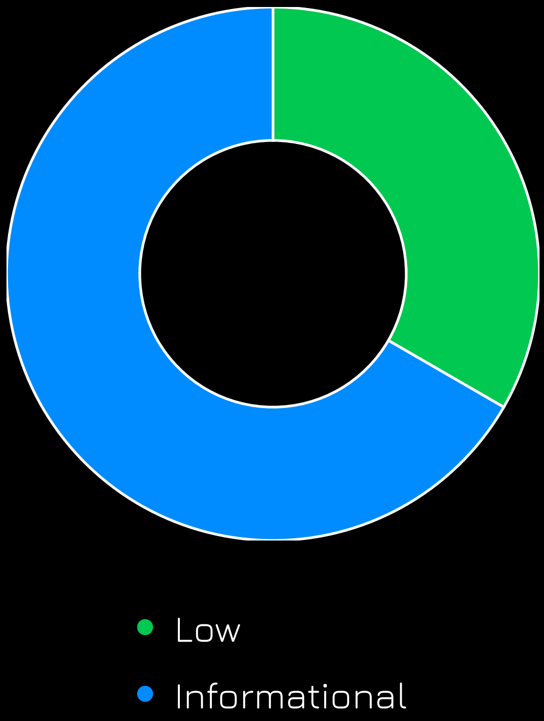
ISSUE SYMBOLIC CODES

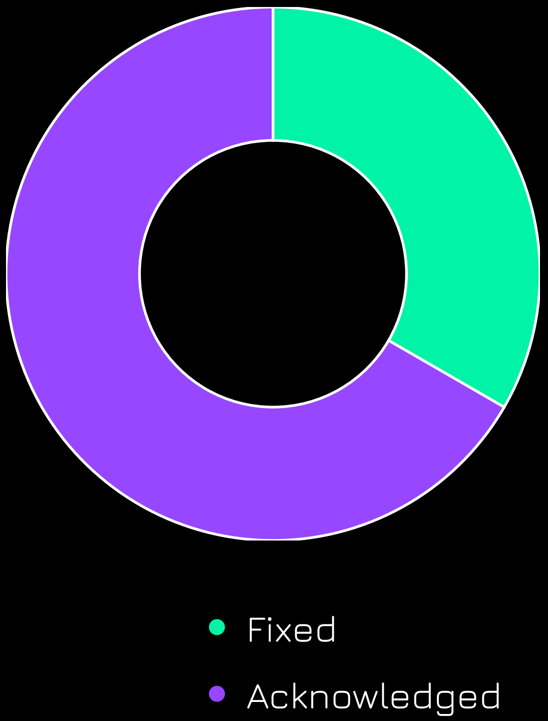
Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

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

Total: 3





WEAKNESSES

This section contains the list of discovered weaknesses.

SPOOL3-4

IMPLEMENT BEFOREDEPOSITCHECK

SEVERITY: Low

PATH:

GearboxV3Strategy.sol:beforeDepositCheck()

REMEDIATION:

Consider adding a suitable implementation for the beforeDepositCheck() function.

STATUS: Fixed

DESCRIPTION:

The Contract **GearboxV3Strategy** contains hooks that facilitate appropriate checks before deposit or redemption. However, the contract FarmingPool imposes a limit on deposits. Consequently, there is a possibility of transaction reverting during **doHardWork()** function execution, resulting in a loss of gas.

```
function beforeDepositCheck(uint256[] memory, uint256[] calldata) public
view override {}
```

```
function deposit(uint256 amount) public virtual {
    _mint(msg.sender, amount);
    if (balanceOf(msg.sender) > _MAX_BALANCE) revert MaxBalanceExceeded();
    stakingToken.safeTransferFrom(msg.sender, address(this), amount);
}
```

SPOOL3-2

REDUNDANT SAFEAPPROVE USAGE

SEVERITY: Informational

PATH:

src/strategies/Strategy.sol:L455-464

REMEDIATION:

Consider changing the call from safeApprove() to approve() to eliminate redundant checks and save gas.

STATUS: Acknowledged

DESCRIPTION:

The **_resetAndapprove()** and **_resetAllowance()** functions have a call to **token.safeApprove()**. However, calling **safeApprove()** is redundant because additional checks have already been conducted before invoking this function.

```
function _resetAndApprove(IERC20 token, address spender, uint256 amount)
internal {
    _resetAllowance(token, spender);
    token.safeApprove(spender, amount);
}

function _resetAllowance(IERC20 token, address spender) internal {
    if (token.allowance(address(this), spender) > 0) {
        token.safeApprove(spender, 0);
    }
}
```

SPOOL3-3

_CALCULATEYIELDPERCENTAGE FUNCTION CAN BE SIMPLIFIED

SEVERITY: Informational

PATH:

src/strategies/Strategy.sol:L443-453

REMEDIATION:

See description

STATUS: Acknowledged

DESCRIPTION:

The body of the **_calculateYieldPercentage()** function can be simplified by using the code provided in the Remediation section.

```
function _calculateYieldPercentage(uint256 previousValue, uint256 currentValue)
    internal
    pure
    returns (int256 yieldPercentage)
{
    if (currentValue > previousValue) {
        yieldPercentage = int256((currentValue - previousValue) * YIELD_FULL_PERCENT / previousValue);
    } else if (previousValue > currentValue) {
        yieldPercentage = -int256((previousValue - currentValue) * YIELD_FULL_PERCENT / previousValue);
    }
}
```

Consider updating the function with the suggested code as it improves gas efficiency.

```
function    _calculateYieldPercentage(uint256    previousValue,    uint256
currentValue)
    internal
    pure
    returns (int256 yieldPercentage)
{
    unchecked {
            yieldPercentage  =  (int(currentValue)  -  int(previousValue))  *
YIELD_FULL_PERCENT_INT / int(previousValue);
    }
}
```

hexens × Spool