hexens × Spool

MAY.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 was focused on the YearnV3 strategy smart contract for the Spool Protocol V2, which supports YearnV3 vaults with gauge and juiced vaults. 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 contracts over a total of 3 days. Throughout this audit, we identified three minor 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/25

https://github.com/SpoolFi/spool-v2-core/pull/28

1. - src/strategies/ERC4626StrategyDouble.sol
   - src/strategies/YearnV3StrategyWithGauge.sol
   - src/strategies/YearnV3StrategyWithJuice.sol

2. - src/access/SpoolAccessControl.sol
   - src/interfaces/ISpoolAccessControl.sol

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

https://github.com/SpoolFi/spool-v2-core/pull/25/commits/bc2e14f4e46dbc56997650fe4bc3416dd9264b52

AUDITING DETAILS

STARTED
27.05.2024

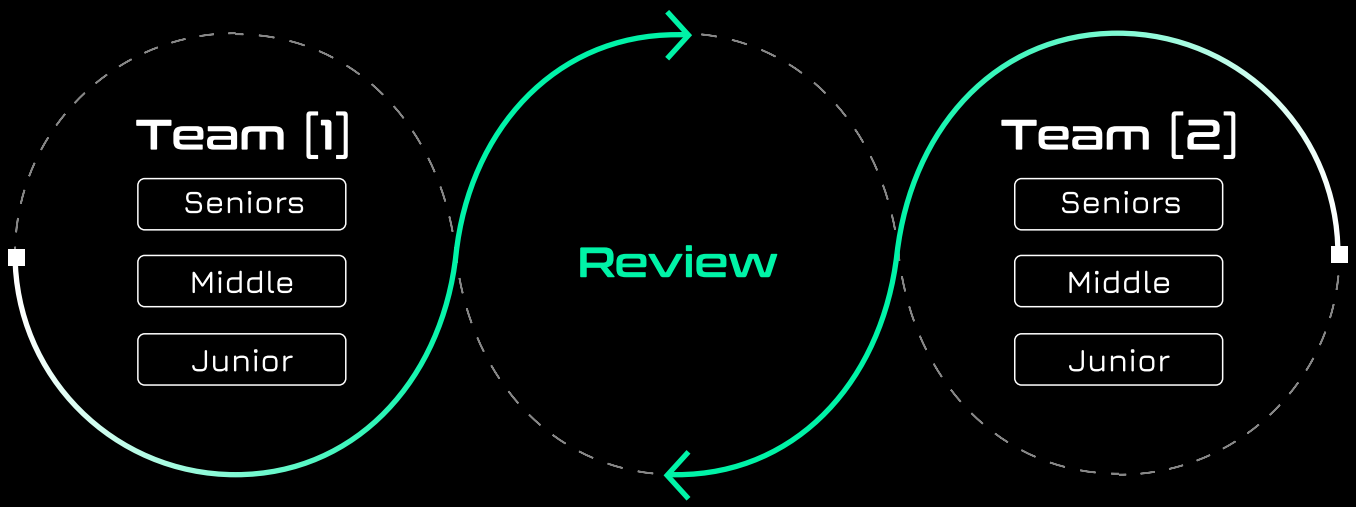
DELIVERED
30.05.2024

Review Led by

MIKHAIL EGOROV
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.

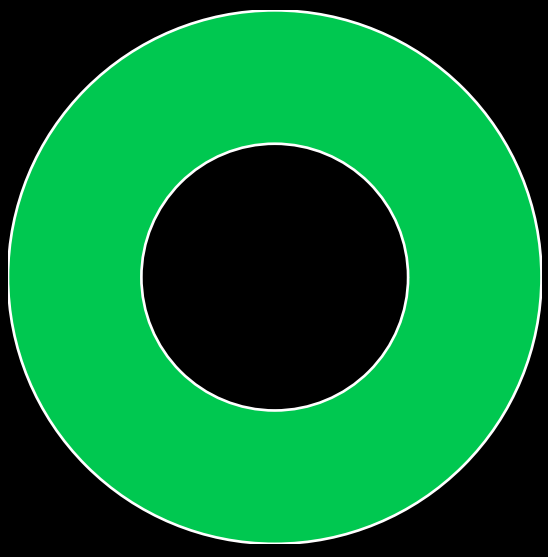
ISSUE SYMBOLIC CODES

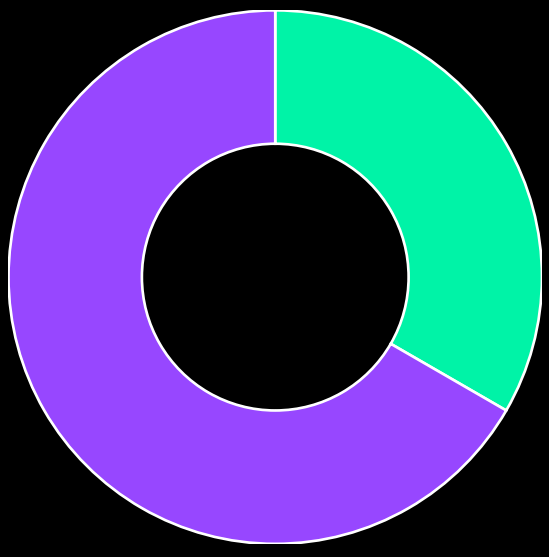
Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

Severity	Number of Findings
Critical	0
High	0
Medium	0
Low	3
Informational	0

Total: 3





WEAKNESSES

This section contains the list of discovered weaknesses.

SPSTR-2

LACK OF USAGE _GETPROTOCOLREWARDSINTERNAL() FUNCTION

SEVERITY: Low

PATH:

src/strategies/YearnV3StrategyWithGauge.sol::_compound():L33-41

REMEDIATION:

Consider updating the _getProtocolRewardsInternal() function by adding this part:

```
if (balance > 0) {
  IERC20(rewards[0]).safeTransfer(address(swapper), balance);
```

Also, use the _getProtocolRewardsInternal() internal function within the _compound() function.

STATUS: Fixed

DESCRIPTION:

Function **_compound()** implements functionality for getting rewards **secondaryVault** and transferring it to the **swapper** contract. It is assumed that the protocol design should facilitate this functionality through the **_getProtocolRewardsInternal** function. Utilizing this internal function could improve gas efficiency and enhance code readability.

1.

```
function _getProtocolRewardsInternal() internal override returns (address[]
memory, uint256[] memory) {
        address[] memory tokens = new address[](1);
        uint256[] memory amounts = new uint256[](1);
        IERC4626 secondaryVault_ = secondaryVault();
        tokens[0] = IYearnGaugeV2(address(secondaryVault_)).REWARD_TOKEN();
        IYearnGaugeV2(address(secondaryVault_)).getReward();
        amounts[0] = IERC20(tokens[0]).balanceOf(address(this));
        return (tokens, amounts);
    }
```

2.

```
function _compound(address[] calldata tokens, SwapInfo[] calldata swapInfo,
uint256[] calldata slippages)//@note tokens[0] is USDC
        internal
        override
        returns (int256 compoundedYieldPercentage)
    {
        if (swapInfo.length == 0) {
            return compoundedYieldPercentage;
        }
        if (slippages[0] > 1) {
            revert CompoundSlippage();
        }
        IERC4626 secondaryVault_ = secondaryVault();
        IYearnGaugeV2(address(secondaryVault_)).getReward();
        address[] memory rewards = new address[](1);
        rewards[0] = IYearnGaugeV2(address(secondaryVault_)).REWARD_TOKEN();
        uint256 balance = IERC20(rewards[0]).balanceOf(address(this));

    }
```

SPSTR-3

BEFOREREDEEMALCHECK_ LACKS ERC4626LIB.ISREDEEMALEMPTY CHECK

SEVERITY: Low

PATH:

src/strategies/ERC4626StrategyDouble.sol:L64-L68
src/strategies/ERC4626StrategyBase.sol:L289-L292

REMEDIATION:

Add ERC4626Lib.isRedeemalEmpty() call within beforeRedeemalCheck_() of ERC4626StrategyDouble to check if the primary vault has enough assets to redeem. Refine the returned value, it should be the amount of assets.

STATUS: Acknowledged

DESCRIPTION:

The **beforeRedeemalCheck_()** function of **ERC4626StrategyDouble** doesn't check if there are enough assets to redeem in the primary vault by calling **ERC4626Lib.isRedeemalEmpty()**. It performs this check only for the secondary vault.

Although the **beforeRedeemalCheck_()** function in **ERC4626StrategyBase** incorporates this check, it gets overridden in the **ERC4626StrategyDouble** parent contract.

Furthermore, the current implementation of **beforeRedeemalCheck_()** returns the amount of shares in the first vault, instead of the amount of assets.

```
function beforeRedeemalCheck_(uint256 shares) internal view virtual override
returns (uint256) {
    IERC4626 secondaryVault_ = secondaryVault();
    if (ERC4626Lib.isRedeemalEmpty(secondaryVault_, shares)) revert
BeforeRedeemalCheck();
    return secondaryVault_.previewRedeem(shares);
}
```

SPSTR-4

FRONT-RUNNING ISSUE WITH OWNERSHIP TRANSFER

SEVERITY: Low

PATH:

src/access/SpoolAccessControl.sol:L66-L70

REMEDIATION:

Inside the transferSmartVaultOwnership() function check that the current owner is not the same as the newOwner. Additionally, implement a timelock to prevent the current owner from calling the transferSmartVaultOwnership() function during a specified period.

STATUS: Acknowledged

DESCRIPTION:

The **transferSmartVaultOwnership()** function in **SpoolAccessControl** doesn't verify that the current smart vault owner is not the same as the **newOwner**.

Consequently, this could result in a front-running issue. In this scenario, the current owner could transfer vault ownership to another entity. Before the new owner accepts the ownership with **acceptSmartVaultOwnership()**, the old owner could reclaim ownership by using **transferSmartVaultOwnership()** and specifying their address as the **newOwner**.

```
function transferSmartVaultOwnership(address smartVault, address newOwner)
external {
    if (msg.sender != smartVaultOwner[smartVault]) revert
OwnableUnauthorizedAccount(msg.sender);
    smartVaultOwnerPending[smartVault] = newOwner;
    emit SmartVaultOwnershipTransferStarted(msg.sender, newOwner);
}
```


hexens
×
Spool