hexens × Spool

APR.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 focused on the new Spool V2 strategies for Arbitrum, including Aave (V3), Aave (V3) swap, Compound (V3), Compound (V3) swap, and Gamma-Camelot strategies.

Our security assessment involved a comprehensive review of the strategy smart contracts, spanning a total of 1 week. During this audit, we identified one high-severity issue, four medium-severity issues, and three low-to-informational-severity issues.

All of our reported issues were either 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/22/commits/997bb13eb78ac731f419fef7deb2bacf24409a93

The issues described in this report were fixed. Corresponding commits are mentioned in the description.

AUDITING DETAILS

STARTED
22.04.2024

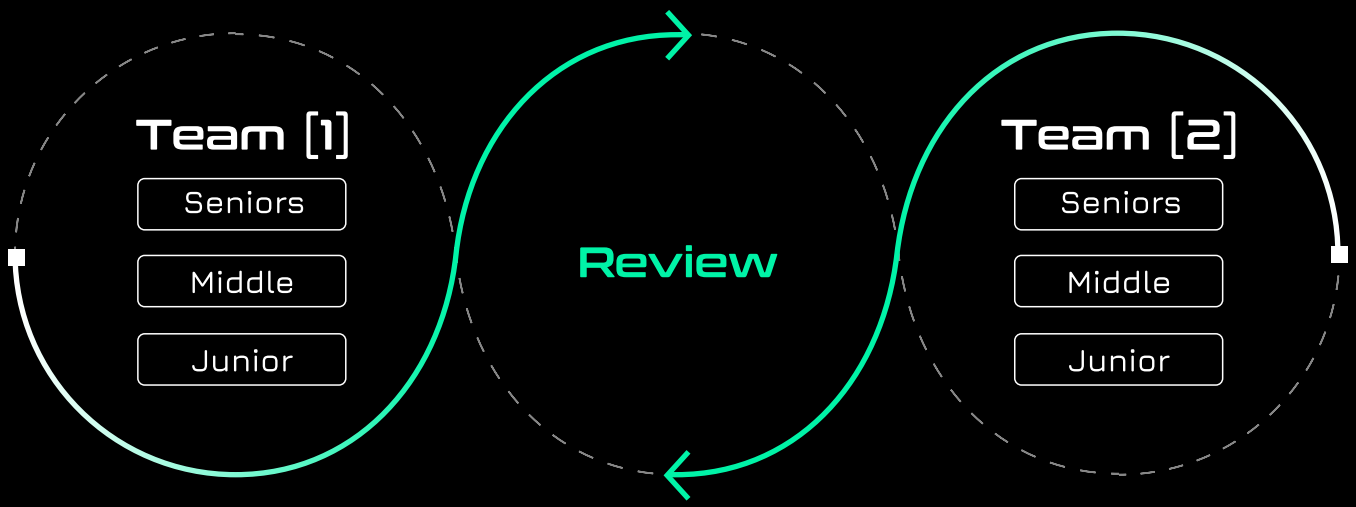
DELIVERED
26.04.2024

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.

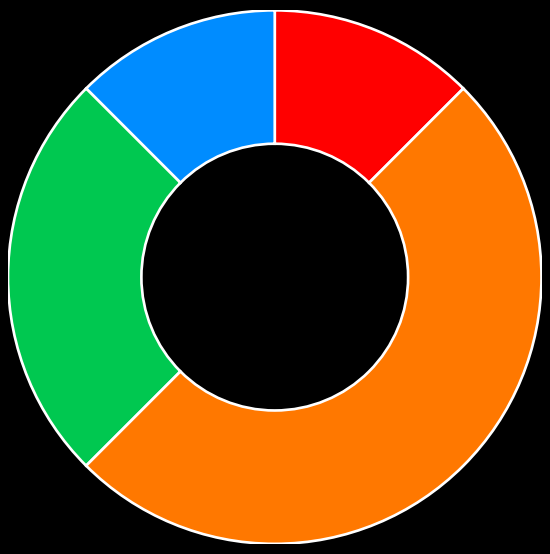
ISSUE SYMBOLIC CODES

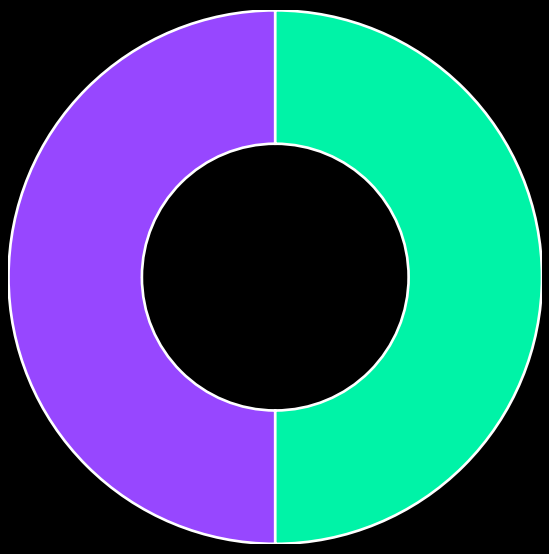
Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

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

Total: 8





WEAKNESSES

This section contains the list of discovered weaknesses.

SPOOL4-1

SHARE PRICE COULD BE MANIPULATED

SEVERITY: High

PATH:

src/strategies/arbitrum/GammaCamelotStrategy.sol:L432-442

REMEDIATION:

Avoid using the spot price from the WETH/USDC Algebra pool as it is susceptible to manipulation. Use price feeds instead.

STATUS: Fixed

DESCRIPTION:

The **_getHypervisorSharePrice()** function calculates the share price for the hypervisor, priced in token1 (USDC) and multiplied by a precision factor of **10**10**18**.

```
function _getHypervisorSharePrice() private view returns (uint256
hypervisorSharePrice) {
    uint256 precision = pool.PRECISION();
    IAlgebraPool underlyingPool = IAlgebraPool(pool.pool());

    (uint160 sqrtPriceX96,,,,,,,) = underlyingPool.globalState();
    uint256 price = FullMath.mulDiv(uint256(sqrtPriceX96) *
uint256(sqrtPriceX96), precision, 2 ** 192);
    (uint256 amount0, uint256 amount1) = pool.getTotalAmounts();
    uint256 amount0PricedInAmount1 = (amount0 * price) / precision;

    hypervisorSharePrice = ((amount0PricedInAmount1 + amount1) * 1 ether) /
(pool.totalSupply());
}
```

When depositing WETH and USDC tokens in the Hypervisor by calling **gammaUniProxy.deposit()**, it creates a position in the Algebra WETH/USDC pool (UniswapV3 fork) between ticks **baseLower** and **baseUpper**.

In the **_getHypervisorSharePrice()** function, it utilizes the spot price (from slot0/globalState) to obtain the WETH price in USDC and stores it in the **amount0PricedInAmount1** variable.

However, the spot price of WETH in the pool could be manipulated, leading to a manipulated share price.

Consequently, an incorrect **baseYieldPercentage** is returned from the **_getYieldPercentage()**. This results in the wrong amount of fees being collected during the execution of the do hard work (DHW) routine, and an incorrect amount of SVT being minted.

```
// Compound and get USD value.
{
    dhwInfo.yieldPercentage = _getYieldPercentage(dhwParams.baseYield);
    int256 compoundYield = _compound(dhwParams.assetGroup,
dhwParams.compoundSwapInfo, dhwParams.slippages);
    dhwInfo.yieldPercentage += compoundYield + compoundYield *
dhwInfo.yieldPercentage / YIELD_FULL_PERCENT_INT;
}

// collect fees, mint SVTs relative to the yield generated
_collectPlatformFees(dhwInfo.yieldPercentage, dhwParams.platformFees);
```

MISSING SLIPPAGE PROTECTION IN THE _GETYIELDPERCENTAGE FUNCTION

SEVERITY: Medium

PATH:

src/strategies/arbitrum/GammaCamelotStrategy.sol::_getHypervisorSharePrice():L432-L442

REMEDIATION:

Consider adding a slippage protection mechanism in the mentioned flow.

STATUS: Fixed

DESCRIPTION:

The **_getYieldPercentage()** function in the **GammaCamelotStrategy.sol** contract calculates the current share price based on the **_getHypervisorSharePrice()** function:

```
function _getYieldPercentage(int256) internal override returns (int256
baseYieldPercentage) {
    uint256 sharePriceCurrent = _getHypervisorSharePrice();

    baseYieldPercentage = _calculateYieldPercentage(_lastSharePrice,
sharePriceCurrent);
    _lastSharePrice = sharePriceCurrent;
}
```

Which retrieves the share price from the underlying Algebra WETH/USDC pool. However, the share price obtained from **_getHypervisorSharePrice()** may be subject to potential slippage, as it effectively performs a swap between USDC and WETH. As a result, if the **sqrtPriceX96** is sandwiched (frontrunned), the attacker can manipulate the amount of **SVT** to be minted and fees collected in DHW.

```
function _getHypervisorSharePrice() private view returns (uint256
hypervisorSharePrice) {
    uint256 precision = pool.PRECISION();
    IAlgebraPool underlyingPool = IAlgebraPool(pool.pool());

    (uint160 sqrtPriceX96,,,,,,,) = underlyingPool.globalState();
    uint256 price = FullMath.mulDiv(uint256(sqrtPriceX96) *
uint256(sqrtPriceX96), precision, 2 ** 192);
    (uint256 amount0, uint256 amount1) = pool.getTotalAmounts();
    uint256 amount0PricedInAmount1 = (amount0 * price) / precision;

    hypervisorSharePrice = ((amount0PricedInAmount1 + amount1) * 1 ether) /
(pool.totalSupply());
}
```

USAGE OF HARDCODED FEE FOR UNIV3 POOL

SEVERITY: Medium

PATH:

src/strategies/helpers/AssetGroupSwapHelper.sol:L54-56

REMEDIATION:

Consider making the fee adjustable and pass it to _depositToProtocol() and _redeemFromProtocol() within the slippages array.

STATUS: Fixed

DESCRIPTION:

The swap strategies **CompoundV3SwapStrategy** and **AaveV3SwapStrategy** execute swaps between input or output tokens and the underlying token of the strategies using the **_assetGroupSwap** function of **AssetGroupSwapHelper**.

However, the **AssetGroupSwapHelper** uses a hardcoded fee value of 100:

```
function _getFee() internal virtual returns (uint24) {
    return 100;
}
```

Consequently, if there are no UniswapV3 pools exist for the input (output) and underlying token with a fee of 100, the call to **_assetGroupSwap** will revert.

USAGE OF HARDCODED LOCKDURATION MAY LEAD TO LOSS OF EXTRA REWARDS

SEVERITY: Medium

PATH:

src/strategies/arbitrum/GammaCamelotStrategy.sol::_getHypervisorSharePrice():L313-333

REMEDIATION:

Consider making the lockDuration adjustable and passing it to createPosition().

STATUS: Acknowledged, see commentary

DESCRIPTION:

In the **GammaCamelotStrategy.sol** contract, the **_depositToProtocolInternal()** function deposits (WETH, USDC) tokens, acquires shares and deposits them in the nftPool by invoking **createPosition()**. For this function, the **lockDuration** argument is hardcoded to 0, which eliminates the potential for additional rewards.

1.

```
function _depositToProtocolInternal(address[] memory tokens, uint256[] memory
amounts, uint256 slippage)
        internal
        returns (uint256 shares)
    {
        if (amounts[0] == 0 && amounts[1] == 0) {
            return 0;
        }
        _resetAndApprove(IERC20(tokens[0]), address(pool), amounts[0]);//@note
spender is pool
        _resetAndApprove(IERC20(tokens[1]), address(pool), amounts[1]);
        uint256[4] memory minAmounts;
        //@note to: address(this)[tokens will be minted in address(this)] //@note
pos: address(pool)
        gammaUniProxy.deposit(amounts[0], amounts[1], address(this),
address(pool), minAmounts);
        shares = pool.balanceOf(address(this));

        if (shares < slippage) {
            revert GammaCamelotDepositSlippagesFailed();
        }

        _resetAndApprove(IERC20(address(pool)), address(nftPool), shares); //
@note spender is nftPool
        // initial deposit
        if (nftId == type(uint256).max) {//@note nftId is type(uint256).max in
constructor
            nftPool.createPosition(shares, 0);//@audit why always lock with 0
lockDuration time  //@note Create a staking position (spNFT) with an optional
lockDuration
            nftId = nftPool.lastTokenId();
            // onERC721Received executed on transfer, which gives allowance to
this contract to use the NFT.
            nftPool.safeTransferFrom(address(this), address(nitroPool), nftId);//
@note transfer nft to nitroPool
        } else {
        /.../
```

2.

```
/.../
// calculate bonuses
uint256 lockMultiplier = getMultiplierByLockDuration(lockDuration);
uint256 amountWithMultiplier = amount.mul(lockMultiplier.add(1e4)).div(1e4);

// create position
_stakingPositions[currentTokenId] = StakingPosition({
  amount : amount,
  rewardDebt : amountWithMultiplier.mul(_accRewardsPerShare).div(1e18),
  lockDuration : lockDuration,
  startLockTime : _currentBlockTimestamp(),
  lockMultiplier : lockMultiplier,
  amountWithMultiplier : amountWithMultiplier,
  boostPoints : 0,
  totalMultiplier : lockMultiplier,
  pendingGrailRewards: 0,
  pendingXGrailRewards: 0
});
/.../
```

Commentary from the client:

- “Acknowledged that potential rewards are being left, but due to the architecture of the Spool V2 protocol, we cannot have funds locked, in case of a liquidity shortage on withdraw.
- Therefore, we will choose to take no further action on this issue.”

LACK OF SLIPPAGE PROTECTION IN _COMPOUND

SEVERITY: Medium

PATH:

src/strategies/arbitrum/base/CompoundV3StrategyBase.sol:L109-133
src/strategies/arbitrum/base/AaveV3StrategyBase.sol:L111-130

REMEDIATION:

Add slippage protection to the _compound routine.

STATUS: Acknowledged, see commentary

DESCRIPTION:

The AaveV3 and CompoundV3 strategies currently do not have slippage protection when swapping reward tokens to strategy tokens within the **_compound** routine.

Furthermore, based on e2e tests, SwapInfo struct also doesn't include any slippage values:

```
SwapInfo[] memory compoundSwapInfo = new SwapInfo[](1);
compoundSwapInfo[0] = SwapInfo({
    swapTarget: address(exchange),
    token: address(compToken),
    swapCallData: abi.encodeCall(exchange.swap, (address(compToken),
compAmount[0], address(swapper)))
});
```

```
function _compound(address[] calldata, SwapInfo[] calldata swapInfo, uint256[]
calldata)
    internal
    override
    returns (int256 compoundedYieldPercentage)
{
    if (swapInfo.length > 0) {
        uint256 compBalance = _getCompoundReward();

        if (compBalance > 0) {
            comp.safeTransfer(address(swapper), compBalance);
            address[] memory tokensIn = new address[](1);
            tokensIn[0] = address(comp);
            address[] memory tokensOut = new address[](1);
            tokensOut[0] = underlying;
            uint256 swappedAmount = swapper.swap(tokensIn, swapInfo, tokensOut,
address(this))[0];

            if (swappedAmount > 0) {
                uint256 cTokenBalanceBefore = _getCTokenBalance();
                _depositToProtocolInternal(swappedAmount);

                compoundedYieldPercentage =
_calculateYieldPercentage(cTokenBalanceBefore, _getCTokenBalance());
            }
        }
    }
}
```

Commentary from the client:

- “the slippages array passed to compound is used for the redepositing stage, not for the rewards sold via swap().
- **swapInfo.swapCallData** will contain encoded slippage values in the real case (ie. 1inch). in e2e tests slippages are not used as the exchange is a mock exchange.
- Therefore, we will choose to take no further action on this issue.”

_COMPOUND DOESN'T USE RETURN VALUE OF _GETPROTOCOLREWARDSINTERNAL

SEVERITY: Low

PATH:

src/strategies/arbitrum/GammaCamelotStrategy.sol:L273

REMEDIATION:

Check the reward balances returned from the call to _getProtocolRewardsInternal() and consider exiting early if there are no reward tokens available.

STATUS: Acknowledged, see commentary

DESCRIPTION:

The **_getProtocolRewardsInternal()** function returns two arrays containing reward tokens and their respective balances. However, the **_compound()** function ignores the second return value, containing the array with reward balances. Consequently, the function's flow proceeds with attempting to swap ARB and GRAIL tokens into WETH and USDC without checking the availability of reward tokens.

The **_compound()** function can check the reward token balances and exit early if there are no reward tokens to swap.

```
function _compound(address[] calldata tokens, SwapInfo[] calldata
compoundSwapInfo, uint256[] calldata slippages)
    internal
    override
    returns (int256 compoundYield)
{
    uint256 slippageOffset;
    if (slippages[0] < 2) {
        slippageOffset = 4;
    } else {
        revert GammaCamelotCompoundSlippagesFailed();
    }

    if (nftId == type(uint256).max || compoundSwapInfo.length == 0) {
        return compoundYield;
    }

    (address[] memory rewardTokens,) = _getProtocolRewardsInternal();

    swapper.swap(rewardTokens, compoundSwapInfo, tokens, address(this));
    // handle getting WETH/USDC rewards
    uint256[] memory swapped = new uint256[](2);
    swapped[0] = IERC20(tokens[0]).balanceOf(address(this));
    swapped[1] = IERC20(tokens[1]).balanceOf(address(this));

    uint256 sharesBefore = _getPoolBalance();
    uint256 shares = _depositToProtocolInternal(tokens, swapped,
slippages[slippageOffset]);
    if (_isViewExecution()) {
        emit Slippages(true, shares, "");
    }

    compoundYield = int256(YIELD_FULL_PERCENT * (_getPoolBalance() -
sharesBefore) / sharesBefore);
}
```

Commentary from the client:

- “The external service that executes DoHardWork checks if there is value to compound. If not, it wouldn't populate **compoundSwapInfo**, and thus would exit before **getProtocolRewardsInternal**.
- Therefore, we will choose to take no further action on this issue.”

USING BLOCK.TIMESTAMP AS DEADLINE

SEVERITY: Low

PATH:

src/strategies/helpers/AssetGroupSwapHelper.sol:L47

REMEDIATION:

Consider specifying a particular deadline value instead of block.timestamp by passing the deadline parameter to _depositToProtocol() and _redeemFromProtocol() within the slippages array.

STATUS: Acknowledged, see commentary

DESCRIPTION:

The swap in UniV3 pool currently utilizes **ExactInputSingleParams** with **block.timestamp** as a deadline parameter. This approach is ineffective as setting **block.timestamp** value does not provide adequate deadline protection against long-pending transactions.

```
function _defaultSwap(address tokenIn, address tokenOut, uint256 amount, uint256
slippage)
    internal
    virtual
    returns (uint256)
{
    IERC20(tokenIn).safeApprove(address(swapRouter), amount);
    IV3SwapRouter.ExactInputSingleParams memory params =
IV3SwapRouter.ExactInputSingleParams({
        tokenIn: tokenIn,
        tokenOut: tokenOut,
        fee: _getFee(),
        recipient: address(this),
        deadline: block.timestamp,
        amountIn: amount,
        amountOutMinimum: slippage,
        sqrtPriceLimitX96: 0
    });

    return swapRouter.exactInputSingle(params);
}
```

Commentary from the client:

- “We already set slippage for the swapped amount
- We already have a deadline in the external service used to execute DoHardWork and Re-allocation
- Therefore, we will choose to take no further action on this issue.”

UNUSED IMPORT IN GAMMACAMELOTSTRATEGY

SEVERITY: Informational

PATH:

src/strategies/arbitrum/GammaCamelotStrategy.sol:L8

REMEDIATION:

Remove the unused import.

STATUS: Fixed

DESCRIPTION:

The **GammaCamelotStrategy** contract imports **ICamelotPair.sol** but it is not utilized anywhere in the code.

```
import "../../external/interfaces/strategies/arbitrum/gamma-camelot/
ICamelotPair.sol";
```


hexens
×
Spool