Panoptic Security Review

Auditors

0xjuaan

0xSpearmint

15th November, 2025

Introduction

Obsidian Audits

Obsidian audits is a team of top-ranked security researchers, with a publicly proven track record, specialising in DeFi protocols across EVM chains and Solana.

The team has achieved 10+ top-2 placements in audit competitions, placing 1st in competitions for Wormhole, Pump.fun, Yearn Finance, and many more.

Find out more: obsidianaudits.com

Audit Overview

Panoptic is a a permissionless options trading protocol built on Uniswap v3 & v4.

Panoptic engaged Obsidian Audits to perform a security review on the smart contracts in the `panoptic-labs/panoptic-next-core-private` repo, focusing on the protocol's v2 update. The review was conducted from the 22nd of October to the 5th of November.

Scope

Files in scope

Repo	Files in scope
`panoptic-next-core-private` **Commit hash:** 0efeee0b9b4b87d1152a0fecab3de02b2910dc25	- contracts/PanopticPool.sol - contracts/RiskEngine.sol - contracts/SemiFungiblePositionManager.sol (partial) - contracts/CollateralTracker.sol (partial) - contracts/PanopticFactory.sol (partial) - libraries/PanopticMath.sol (partial) - libraries/Math.sol (partial) For partially scoped files, refer to the changes in PR #183

Summary of Findings

Severity Breakdown

A total of **11** issues were identified and categorized based on severity:

- **2 High severity**
- **1 Medium severity**
- **4 Low severity**
- **4 Informational**

Findings Overview

ID	Title	Severity	Status
H-01	`forceExercise` will not be possible for positions with a credit leg (width==0)	High	Fixed
H-02	`_getRequiredCollateralSingleLegPartner` does not verify that each leg has the same size when reducing collateral requirements	High	Fixed
M-01	Incorrect bit-shift logic for the CollateralTracker's CREATE2 seed	Medium	Fixed
L-01	The `safeTransferFrom` function does not check if the `token` has code	Low	Acknowledged
L-02	An attacker can manipulate the utilisation rate to mint an option at a lower collateralisation ratio	Low	Fixed
L-03	Incorrect rounding in `getAmountsMoved` causes unnecessary reverts	Low	Fixed
L-04	Interest is overcharged in some scenarios, enabling arbitrary amounts of donations to the CollateralTracker	Low	Fixed

ID	Title	Severity	Status
I-01	Incorrect comment in \`PanopticFactory.deployNewPool()\`	Informational	Fixed
I-02	The \`accrueInterestTo\` function is restricted to the \`PanopticPool\`, but is never called by it	Informational	Fixed
I-03	Incorrect comments	Informational	Acknowledged
I-04	Zero-value actions in the CollateralTracker are not completely prevented	Informational	Fixed

Severity Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- **High -** leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium -** leads to a moderate material loss of assets in the protocol or moderately harms a group of users. Alternatively, breaking a core aspect of the protocol's intended functionality
- **Low -** leads to a minor material loss of assets in the protocol or harms a small group of users.

Likelihood

- **High -** attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.

- **Medium -** the attack/vulnerability requires minimal or no preconditions, but there is limited or no incentive to exploit it in practice
- **Low -** requires highly unlikely precondition states, or requires a significant attacker capital with little or no incentive.

Findings

[H-01] `forceExercise` will not be possible for positions with a credit leg (width==0)

Description

The `RiskEngine.exerciseCost()` function is called during a forced exercise of a position. To compute the cost of exercising the position, it iterates over each leg of the position and performs calculations.

The issue is that the new 'credit' leg type (signified by a `width` equal to `0`) can also be a part of this position, and the usage of `width==0` is not accounted for.

Within the loop, `PanopticMath.getLiquidityChunk()`, with the position's leg as a parameter. The issue is that this performs UniV3 math calculations even though the position width is set to 0:

```
return Math.getLiquidityForAmount0(tickLower, tickUpper, amount);
```

This will lead to a revert, due to division by zero in the following calculation, since `highPriceX96 == lowPriceX96`:

```
uint256 liquidity = mulDiv(
    amount0,
    mulDiv96(highPriceX96, lowPriceX96),
    highPriceX96 - lowPriceX96
);
```

The impact is that any position which includes a credit leg cannot be force exercised, due to the revert in `exerciseCost()`.

Recommendation

Consider adding a constraint at the start of the loop in `RiskEngine.exerciseCost()` to ignore legs with `width==0`

```
int24 width = tokenId.width(leg);
+if (width == 0) continue;
```

Remediation: Fixed in PR #195

[H-02] `_getRequiredCollateralSingleLegPartner` does not verify that each leg has the same size when reducing collateral requirements

Description

The `_getRequiredCollateralSingleLegPartner` function calculates the collateral for a leg of a partnered options position. For certain paired positions (synthetic stocks, spreads, strangles), the system currently relaxes collateral requirements based on the characteristics of the partnered legs.

However, the code does not verify that both legs have the same size when applying the relaxed collateral requirements.

For example, a synthetic stock position's (short put + long call or short call + long put) collateral requirement will only depend on the short leg.

```
} else if (_isLong != isLongP) {
    // SYNTHETIC STOCK: different token types, one is long and the other is
short
    return
        // return the collateral requirement of the short leg only, the
long leg is free!
        _isLong == 0
            ? _getRequiredCollateralSingleLegNoPartner(
                tokenId,
                index,
                positionSize,
                atTick,
                poolUtilization
            )
            : 0;
}
```

As a result, a user can open a position with the long call's size being 100x the partner short put's size, and only put up the required collateral for the relatively small short leg. This results in the larger call being collateralised at a significantly lower level.

Recommendation

The fix is to require that each leg has the same size when loosening collateral requirements in `_getRequiredCollateralSingleLegPartner`:

```diff
- if (tokenId.asset(partnerIndex) == tokenId.asset(index)) {
+ if (tokenId.asset(partnerIndex) == tokenId.asset(index) &&
tokenId.optionRatio(partnerIndex) == tokenId.optionRatio(index)) {
```

Additionally, for synthetic stocks, consider also validating that both legs share the same strike.

Remediation: Fixed in PR #203

[M-01] Incorrect bit-shift logic for the CollateralTracker's CREATE2 seed

Description

In `PanopticFactory.deployNewPool()`, the following logic is used to calculate the seed used when creating the `CollateralTracker` contracts:

```
bytes32 salt32 = bytes32(
    abi.encodePacked(
        uint80(uint160(msg.sender) >> 80),
        uint80(uint160(address(v3Pool)) >> 40),
        uint80(uint160(address(riskEngine)) >> 40),
        salt
    )
);
```

With the current code, the final `bytes32` will incorrectly only store 16 bits of the `uint96 salt`, because the first 3 `uint80` elements take up 80 bits. (256 - 3*80 = 16)

The mistake is that while it's intended to use the first 40 bits of the `v3Pool` and `riskEngine` addresses, they are casted to `uint80` instead of `uint40`.

The impact is that due to the truncated `salt` parameter, there are only 16 bits of entropy in calculation of the CREATE2 salt, limiting the ability to mine addresses with leading zeroes.

Also, currently since these two addresses are shifted right by 40 bits (instead of 120), the result does not include the first 40 bits of the address.

Proof of Concept

The following test demonstrates the issue with the current calculation:

```
    function test_salt32() public {

        // arbitrary addresses used for demonstration
        address v3Pool = address(msg.sender);
        address riskEngine =
address(0xCcCCccccCCCCcCCCCCCcCcCccCcCCCcCcccccccC);
        address sender =
address(0xaAaAaAaaAaAaAaaAaAAAAAAAAaaaAaAaAaaAaaAa);
```

```
        uint96 salt = type(uint96).max;

        console.log("v3Pool", v3Pool);

        bytes32 salt32_original = bytes32(
        abi.encodePacked(
            uint80(uint160(sender) >> 80),
            uint80(uint160(address(v3Pool)) >> 40),
            uint80(uint160(address(riskEngine)) >> 40),
            salt
        )
        );

         bytes32 salt32_correct = bytes32(
            abi.encodePacked(
                uint80(uint160(sender) >> 80),
                uint40(uint160(address(v3Pool)) >> 120),
                uint40(uint160(address(riskEngine)) >> 120),
                salt
            )
        );

        console.log("salt32_original: ");
        console2.logBytes32(salt32_original);

        console.log("salt32_correct: ");
        console2.logBytes32(salt32_correct);
    }
```

Console output:

```
[PASS] test_salt32() (gas: 5843)
Logs:
  v3Pool 0x1804c8AB1F12E6bbf3894d4083f33e07309d1f38
  salt32_original:
  0xaaaaaaaaaaaaaaaaaaaa12e6bbf3894d4083f33eccccccccccccccccccccffff
  salt32_correct:
  0xaaaaaaaaaaaaaaaaaaaa1804c8ab1fccccccccccffffffffffffffffffffffff
```

In `salt32_original`, it can be seen that the `salt` (0xfff...f) only appears for 4 characters, even though it's a `uint96`.

Recommendation

Consider applying the following diff to fix the bug:

```diff
bytes32 salt32 = bytes32(
             abi.encodePacked(
                 uint80(uint160(msg.sender) >> 80),
-                uint80(uint160(address(v3Pool)) >> 40),
-                uint80(uint160(address(riskEngine)) >> 40),
+                uint40(uint160(address(v3Pool)) >> 120),
+                uint40(uint160(address(riskEngine)) >> 120),
                 salt
             )
         );
```

Alternatively, for more simplicity and to avoid bitshifting logic, consider hashing together the 4 elements to compute the CREATE2 salt.

Remediation: Fixed in PR #192

[L-01] The `safeTransferFrom` function does not check if the `token` has code

Description

The `SafeTransferLib`'s functions are taken from an older version of solmate, which doesn't check if the `token` is a deployed contract.

When `safeTransferFrom()` is called on an address where a token hasn't been deployed, the call will succeed as a no-op, without reverting.

This behaviour can lead to severe exploits. For example, in the `CollateralTracker`, it uses `safeTransferFrom` to transfer the underlying token from the user to the pool, before minting them shares.

If the `s_underlyingToken` was not yet deployed, a user can call `deposit()` to mint an arbitrary amount of shares without transferring any assets. Then once the token is deployed and other users perform legitimate deposits, the attacker can withdraw their freely minted shares.

This is currently not exploitable because when creating a new panoptic pool, the following line is reached, which reads the `totalSupply()` of `token0` and `token1`, which would revert if they did not have code. Either way, we still recommend applying the fix to reduce the attack surface.

Here is a writeup describing this vulnerability in more detail: https://forum.balancer.fi/t/balancer-v2-token-frontrun-vulnerability-disclosure/6309

Recommendation

It's recommended to update the `SafeTransferLib` to perform a code size check, to ensure that the `token` is a deployed contract.

Here is the latest version of solmate's `safeTransferFrom` function: https://github.com/transmissions11/solmate/blob/89365b880c4f3c786bdd453d4b8e8fe410344a69/src/utils/SafeTransferLib.sol#L30-L58

Remediation: Issue acknowledged

[L-02] An attacker can manipulate the utilisation rate to mint an option at a lower collateralisation ratio

Description

The `sellCollateralRatio` when selling an option depends on the pool's current utilisation

As utilisation increases, the required `sellCollateralRatio` also increases.

When the pool is in a high utilisation state (which will require a high CR when selling an option), an attacker can do the following to mint the option at a significantly lower CR:

1. Use a fee-free flashloan from Morpho or Uni-V4 to deposit a large amount of assets to the `CollateralTracker` contract
2. Step (1) will instantly decrease the utilisation rate
3. Now they can sell an option from a different account at a significantly lower CR, due to the low utilisation
4. After minting the position, they can withdraw the deposited funds and repay the flash loan

Recommendation

Consider charging a fee on withdrawals that occur very soon after the user's deposit to disincentivise such behaviour that temporarily manipulates utilisation rates

Remediation: Fixed in PR #217

[L-03] Incorrect rounding in `getAmountsMoved` causes unnecessary reverts

Description

The `getAmountsMoved()` function calculates the amounts moved when creating a UniV3 position for a specfic `LiquidityChunk`.

When burning options, the `computeExercisedAmounts` function (which calls `getAmountsMoved()`) is called to obtain `longAmounts` and `shortAmounts`.

This rounds UP the token amounts for short positions, even though Uniswap V3 actually rounds down the withdrawn token amounts when burning liquidity.

```
if (tokenId.isLong(legIndex) == 0) {
    amount0 = uint128(Math.getAmount0ForLiquidityUp(liquidityChunk));
    amount1 = uint128(Math.getAmount1ForLiquidityUp(liquidityChunk));
} else {
    amount0 = uint128(Math.getAmount0ForLiquidity(liquidityChunk));
    amount1 = uint128(Math.getAmount1ForLiquidity(liquidityChunk));
}
```

This leads to a mismatch in `swappedAmount` and `shortAmount` in `CollateralTracker::_updateBalancesAndSettle()` when closing short positions. Assuming there were no ITM amounts, `swappedAmount` will be `1` less than `shortAmount` due to the difference in rounding. This will cause 1 wei of the CollateralTracker shares to be burned from the option owner, which will revert if the user's position was solely collateralised by shares of the alternate token.

Recommendation

Consider using an adapted version of the function, which accounts for whether an option is being minted or burned. If being burned, then token amounts for short positions should be rounded down, and vice versa for long positions.

Remediation: Fixed in PR #208

[L-04] Interest is overcharged in some scenarios, enabling arbitrary amounts of donations to the CollateralTracker

Description

For interest accrual in the `CollateralTracker`, if the interest shares to burn exceed the user's share balance, their balance is burned and the `userBorrowIndex` is set to it's original value.

```
if (shares > userBalance) {
    // update the accrual of interest paid
    burntInterestValue = Math
        .mulDiv(userBalance, _totalAssets, totalSupply)
        .toUint128();
    /// Insolvent case: Pay what you can
    _burn(_owner, userBalance);


    /// @dev DO NOT update index. By keeping the user's old baseIndex,
their debt continues to compound correctly from the original point in time.
    userBorrowIndex = userState.rightSlot();
```

The intention is to ensure that the user's liabilities are persistent since they have not fully paid off their interest.

The issue with not updating the user's borrow index is that this doesn't account for the interest that they have paid (by burning their shares). So in the future when the user tops up their balance and the interest is paid, they will pay the entire obligation again, as if the previous repay never happened.

Recommendation

Consider using a mapping to store the 'pendingInterestShares' for each user. Whenever interest is accrued for the user and their balance cannot cover the owed interest, burn their balance and record the user's shortfall in this mapping. Then, let the user's borrow index be updated to the latest borrow index.

In the future, whenever accruing interest, if the 'pendingInterestShares' of the user is non-zero, this should be paid off before attempting to pay their newly accrued interest. This ensures that interest is not overcharged.

Remediation: Fixed in PR #215

[I-01] Incorrect comment in `PanopticFactory.deployNewPool()`

Description

The comment regarding the CollateralTrackers' CREATE2 salt format is as follows:

```
// salt format: (first 20 characters of deployer address) + (first 20
characters of UniswapV3Pool) + (uint96 user supplied salt)
```

This should be updated to include the `riskEngine`, and the size of the `UniswapV3Pool`'s component should be reduced.

Remediation: Fixed in PR #192

[I-02] The `accrueInterestTo` function is restricted to the `PanopticPool`, but is never called by it

Description

The `accrueInterestTo()` function in the `CollateralTracker` is restricted by the `onlyPanopticPool` modifier.

However, the `PanopticPool` contract does not include code to call `accrueInterestTo`, so this function is not callable.

Remediation: Fixed in PR #194

[I-03] Incorrect comments

Description

This issue aggregates errors in code comments.

1. In `_payCommissionAndWriteData()`, the `uint32 utilizations` return value is a packing of two 16-bit values. However the comment states that it's two 64-bit values
2. This comment is outdated because `_checkSolvencyAtTicks()` no longer reverts if insolvent at a tick.
3. Comments (1, 2, 3) regarding the `_updatePositionsHash()` function still reference the

previous method of calculating position fingerprints (XOR of hashes)

Remediation: Issue acknowledged

[I-04] Zero-value actions in the CollateralTracker are not completely prevented

Description

PR #161 was introduced to prevent zero-value deposits/withdrawals to limit the reachability of unwanted states.

Currently, zero-asset actions are only prevent in the `mint()`, `redeem()`, and overloaded `withdraw()` function, but are not prevented in the `ERC4626.deposit` and `ERC4626::withdraw` functions.

In addition, it's recommended to also prevent actions which lead to zero shares being minted/burned, since these shouldn't occur in the expected usage of the contract.

Remediation: Fixed in PR #204