4th05

Plaza Finance

Security Review Report

23 January 2025

Security Review Report

4th05

January 23, 2025

Table of Contents

Protocol Summary

Plaza is a platform for programmable derivatives built as a set of Solidity smart contracts on Base. It offers two core products: bondETH and levETH, which are programmable derivatives of a pool of ETH liquid staking derivatives (LSTs) and liquid restaking derivatives (LRTs) such as wstETH. Users can deposit an underlying pool asset like wstETH and receive levETH or bondETH in return, which are represented as ERC20 tokens. These tokens are composable with protocols such as DEXes, lending markets, restaking platforms, etc.

bondETH and levETH represent splits of the total return of the underlying pool of ETH LSTs and LRTs, giving users access to a profile of risk and returns that better suits their needs and investment style. Plaza operates in a fully permissionless manner, with each core function of the protocol executable by anyone.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

Overview

Contest platform	Sherlock
LOC	1602
Language	Solidity
Commit	14a962c52a8f4731bbe4655a2f6d0d85e144c7c2
Previous audits	Zellic audit

Scope

- Auction.sol
- BalancerOracleAdapter.sol
- BalancerRouter.sol
- BondOracleAdapter.sol
- BondToken.sol
- Distributor.sol
- LeverageToken.sol
- OracleFeeds.sol
- OracleReader.sol
- Pool.sol
- PoolFactory.sol
- PreDeposit.sol
- Decimals.sol
- ERC20Extensions.sol
- Utils.sol
- Deployer.sol

Issues found

Severity	Number of issues found
High	1
Medium	0
Low	1
Info	0

Findings

[H1] Wrong period considered in `Pool::transferReserveToAuction`

Summary

The `Pool::transferReserveToAuction` function uses a wrong period to transfer the `reserveToken` amount to the auction. The proper period is the `currentPeriod-1` while instead the function uses the `currentPeriod`.

Relevant GitHub Links

https://github.com/sherlock-audit/2024-12-plaza-finance-0x4th05/blob/main/plaza-evm/src/Pool.sol#L578-L579

https://github.com/sherlock-audit/2024-12-plaza-finance-0x4th05/blob/main/plaza-evm/src/BondToken.sol#L217-L229

https://github.com/sherlock-audit/2024-12-plaza-finance-0x4th05/blob/main/plaza-evm/src/Pool.sol#L567

Root Cause

The proper period to be used is the `currentPeriod-1` because when creating an auction the current period in the bond contract increases by 1 cause of the function called `BondToken::increaseIndexedAssetPeriod`. So as it is every time the `Pool::transferReserveToAuction` function is called it will always get the `address(0)` as the `auctions[currentPeriod]`.

```
function startAuction() external whenNotPaused() {
    // Check if distribution period has passed
    require(lastDistribution + distributionPeriod < block.timestamp,
        DistributionPeriodNotPassed());

    // Check if auction period hasn't passed
    require(lastDistribution + distributionPeriod + auctionPeriod >=
        block.timestamp, AuctionPeriodPassed());

    // Check if auction for current period has already started
    (uint256 currentPeriod,) = bondToken.globalPool();
    require(auctions[currentPeriod] == address(0),
        AuctionAlreadyStarted());

    uint8 bondDecimals = bondToken.decimals();
    uint8 sharesDecimals = bondToken.SHARES_DECIMALS();
    uint8 maxDecimals = bondDecimals > sharesDecimals ? bondDecimals :
        sharesDecimals;

    uint256 normalizedTotalSupply = bondToken.totalSupply().
        normalizeAmount(bondDecimals, maxDecimals);
    uint256 normalizedShares = sharesPerToken.normalizeAmount(
        sharesDecimals, maxDecimals);

    // Calculate the coupon amount to distribute
    uint256 couponAmountToDistribute = (normalizedTotalSupply *
        normalizedShares)
         .toBaseUnit(maxDecimals * 2 - IERC20(couponToken).safeDecimals
            ());

    auctions[currentPeriod] = Utils.deploy(
      address(new Auction()),
      abi.encodeWithSelector(
        Auction.initialize.selector,
        address(couponToken),
        address(reserveToken),
        couponAmountToDistribute,
        block.timestamp + auctionPeriod,
        1000,
        address(this),
        poolSaleLimit
      )
    );

    // Increase the bond token period
 @> bondToken.increaseIndexedAssetPeriod(sharesPerToken);

    // Update last distribution time
    lastDistribution = block.timestamp;
  }
```

```
  function increaseIndexedAssetPeriod(uint256 sharesPerToken) public
      onlyRole(DISTRIBUTOR_ROLE) whenNotPaused() {
    globalPool.previousPoolAmounts.push(
      PoolAmount({
        period: globalPool.currentPeriod,
        amount: totalSupply(),
        sharesPerToken: globalPool.sharesPerToken
      })
    );
@>  globalPool.currentPeriod++;
    globalPool.sharesPerToken = sharesPerToken;

    emit IncreasedAssetPeriod(globalPool.currentPeriod, sharesPerToken)
        ;
  }
```

```
function transferReserveToAuction(uint256 amount) external virtual {
@>  (uint256 currentPeriod, ) = bondToken.globalPool();
@> address auctionAddress = auctions[currentPeriod];
  require(msg.sender == auctionAddress, CallerIsNotAuction());

  IERC20(reserveToken).safeTransfer(msg.sender, amount);
}
```

Internal Pre-conditions

An auction is created and ends with the sate `SUCCEEDED`.

Attack Path

The `auction` contract will try to call the `Pool::transferReserveToAuction` but it will not get the `reserveToken` amount because of the `require`. This because of the wrong period used in `Pool::transferReserveToAuction`.

```
function transferReserveToAuction(uint256 amount) external virtual {
 @>   (uint256 currentPeriod, ) = bondToken.globalPool();
 @>   address auctionAddress = auctions[currentPeriod];
 @>   require(msg.sender == auctionAddress, CallerIsNotAuction());
      IERC20(reserveToken).safeTransfer(msg.sender, amount);
  }
```

Impact

Every auction that ends with the state `SUCCEEDED` will not be able to get the amount of the `reserveToken` it should.

Mitigation

```
function transferReserveToAuction(uint256 amount) external virtual {
      (uint256 currentPeriod, ) = bondToken.globalPool();
-     address auctionAddress = auctions[currentPeriod];
+     uint256 previousPeriod = currentPeriod - 1;
+     address auctionAddress = auctions[previousPeriod];
      require(msg.sender == auctionAddress, CallerIsNotAuction());
      IERC20(reserveToken).safeTransfer(msg.sender, amount);
  }
```

[L1] Wrong implementation of the `Pool::NotInAuction`

Summary

The `NotInAuction` modifier does not work as it should because the `auctions[currentPeriod]` will always be the `address(0)`. This because every time a new auction starts the `currentPeriod` gets +1.

Relevant GitHub Links

https://github.com/sherlock-audit/2024-12-plaza-finance-0x4th05/blob/main/plaza-evm/src/Pool.sol#L750C2-L754C4

Root Cause

The `NotInAuction` modifier checks something that it will be always verified being in this way useless. Every time an auction is created the `BondToken::increaseIndexedAssetPeriod` function increases the `currentPeriod`. Therefore, the condition `require(auctions[currentPeriod] == address(0)` is always verified whatever it is the `currentPeriod` considered.

Internal Pre-conditions

An auction is created.

Attack Path

Parameters like `AuctionPeriod`, `DistributionPeriod`, `SharesPerToken`, are changed relying on the `NotInAuction` which however does not work as it should allowing all the parameters to be changed when they should be not.

Impact

Although `GOV_ROLE` role is trusted (trust inputs), it will rely on the `NotInAuction` modifier (otherwise no need to even write it) when changing some parameters using the functions called: `setSharesPerToken`, `setAuctionPeriod`, `setDistributionPeriod`.

Any change made on these parameters during an ongoing auction could have a huge impact on all the users.

Mitigation

Depending on what is the exact moment to check, some solutions could be:

```
  modifier NotInAuction() {
    (uint256 currentPeriod,) = bondToken.globalPool();
-   require(auctions[currentPeriod] == address(0), AuctionIsOngoing());
+   require (block.timestamp > lastdistribution + distributionPeriod +
    auctionperiod, AuctionIsOngoing())
    _;
  }
```

```
  modifier NotInAuction() {
    (uint256 currentPeriod,) = bondToken.globalPool();
-   require(auctions[currentPeriod] == address(0), AuctionIsOngoing());
+   previousPeriod = currentPeriod-1;
+   require(block.timestamp > auctions[previousPeriod].endTime())
    _;
  }
```