Meta Pool Exploit Post-mortem

Background

Meta Pool is a multi-chain liquid staking protocol. Its Ethereum-based staking product issues a share token called `mpETH`, allowing users to stake ETH and receive a transferable, redeemable token that reflects their stake. The `Staking` contract inherits from `ERC4626`, which provides standardized deposit, withdraw, mint, and redeem interfaces.

Attack Summary

On **June 17, 2025**, an attacker exploited the `mint()` function in the `Staking` contract to mint `mpETH` without providing any ETH. This resulted in the protocol losing approximately **56.35 ETH** worth of assets across Ethereum and bridged L2 chains (i.e., **Optimism** and **Linea**). The first attack was front-run and executed by an MEV bot in tx, which then attempted to redeem and bridge the minted `mpETH`. A second exploit occurred shortly after, following the same strategy in a separate tx.

Root Cause◊

The `Staking` contract inherits `mint()` from OpenZeppelin's `ERC4626Upgradeable`, which is a `public` function. The protocol failed to override or restrict this function. Since `mint()` internally calls `_deposit()` without any value transfer checks, a malicious user could mint arbitrary `mpETH` without sending ETH, violating the token's 1:1 backing.

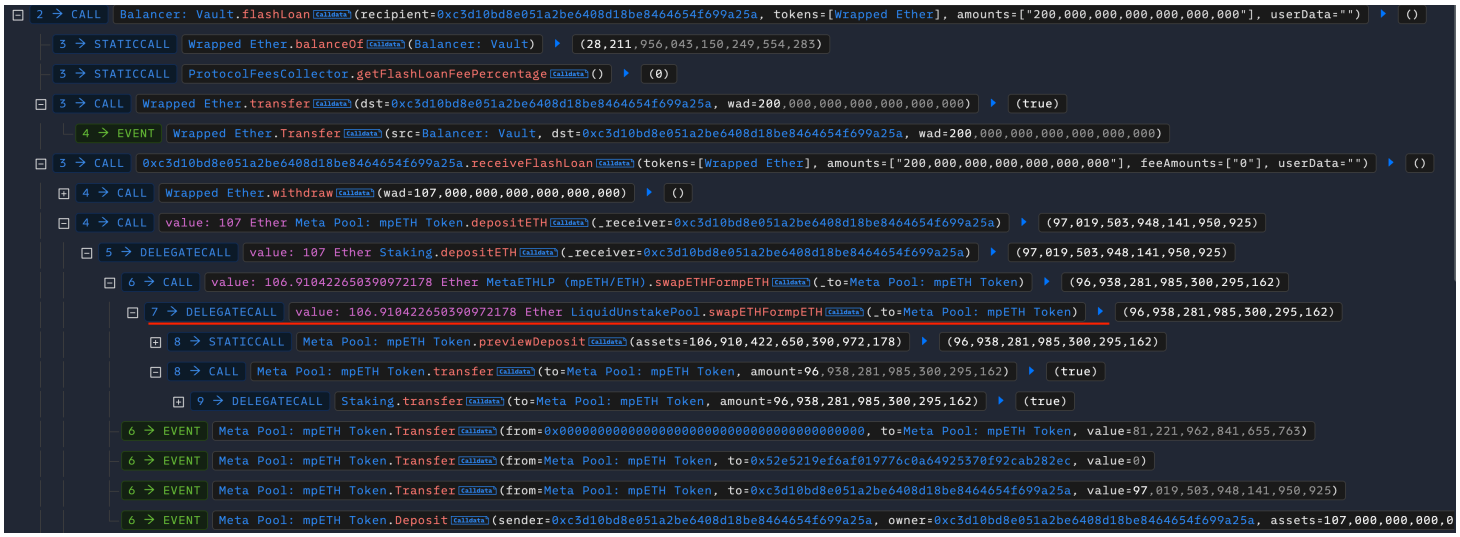
```
183         /// @inheritdoc IERC4626
184         function mint(uint256 shares, address receiver) public virtual returns (uint256) {
185             uint256 maxShares = maxMint(receiver);
186             if (shares > maxShares) {
187                 revert ERC4626ExceededMaxMint(receiver, shares, maxShares);
188             }
189
190             uint256 assets = previewMint(shares);
191             _deposit(_msgSender(), receiver, assets, shares);
192
193             return assets;
194         }
195
```

```
355         function _deposit(
356             address _caller,
357             address _receiver,
358             uint256 _assets,
359             uint256 _shares
360         ) internal override checkWhitelisting {
361             if (_assets < MIN_DEPOSIT) revert DepositTooLow(MIN_DEPOSIT, _assets);
362             (uint256 sharesFromPool, uint256 assetsToPool) = _getmpETHFromPool(_shares, address(this));
363             uint256 sharesToMint = _shares - sharesFromPool;
364             uint256 assetsToAdd = _assets - assetsToPool;
365 
366             if (sharesToMint > 0) _mint(address(this), sharesToMint);
367             totalUnderlying += assetsToAdd;
368 
369             uint256 sharesToUser = _shares;
370 
371             if (msg.sender != liquidUnstakePool) {
372                 uint256 sharesToTreasury = (_shares * depositFee) / 10000;
373                 _transfer(address(this), treasury, sharesToTreasury);
374                 sharesToUser -= sharesToTreasury;
375             }
376 
377             _transfer(address(this), _receiver, sharesToUser);
378 
379             emit Deposit(_caller, _receiver, _assets, _shares);
380         }
```

Attack Flow

Take the attack tx as an example:

1. The attacker flashloaned 200 `WETH` from Balancer, withdrew 107 `WETH` as `ETH`, and used it to invoke the `depositETH()` function of the `mpETH` contract.
   This step was intended to swapped out all `mpETH` from the internal liquidity pool (`liquidUnstakePool`).
   This is crucial because the `mint()` function internally calls `_deposit()`, which further invokes `_getMpETHFromPool()`.
   If there is any `mpETH` in the pool, the contract will attempt to swap attached `ETH` for `mpETH`.
   However, since `mint()` is not payable and the contract does not hold `ETH`, such a swap would fail.
   Therefore, the attacker emptied the pool in advance to ensure the swap is skipped during minting.



2. The attacker then invoked the `mint()` function directly, which minted approximately 9701 `mpETH` without providing any `ETH`, due to the missing override or restriction on the inherited `ERC4626` `mint()` function.
3. Finally, the attacker swapped part of the minted mpETH for ETH, repaid the flashloan, and kept the remaining ETH as profit.

Timeline

Time (UTC)	Event
2025-06-17 06:18	First exploit front-run by **MEV bot**: tx1 - 45.79 ETH gained
2025-06-17 07:08	First manual exploit (tx a): tx2 – 1.64 ETH gained
2025-06-17 07:10	Second manual exploit (tx b): tx3 – 0.03 ETH gained
2025-06-17 08:02	Flashloan-based exploit (tx c): tx4 – 8.89 ETH gained and ~9700 `mpETH` minted
2025-06-17 08:25	BlockSec team **notified and provided the solution to Meta Pool**
2025-06-17 11:05	Meta Pool **disabled mpETH transfer functionality** by upgrading contract
2025-06-18 onward	Postmortem analysis, fund tracking, and remediation measures continued

Meanwhile, a small amount of `mpETH` was discovered to have bridged to L2s (i.e., **Optimism** and **Linea**). Due to the limited liquidity on these chains, the overall impact was assessed to be minimal.

Before the `transfer` function was disabled, the attacker completed the following three actions:

1. **Bridged mpETH to other chains**
   The attacker bridged the exploited `mpETH` to Layer 2 networks such as Optimism and Linea. While liquidity on these chains was limited and no further loss occurred, the bridged `mpETH` must still be handled in coordination with the respective bridge providers.
2. **Redeemed mpETH for native ETH via MetaPool**
   `mpETH` functions as a share token and can be redeemed in MetaPool for native ETH. The attacker used this mechanism to burn `mpETH` and initiate withdrawals. Fortunately, native ETH release is controlled by a privileged role, so the actual funds remained secure.
3. **Added single-sided liquidity to Uniswap V3 pool**
   The attacker deposited `mpETH` as one-sided liquidity into a Uniswap V3 pool. To prevent further users from interacting with this compromised liquidity, the pool should be marked as **blacklisted**.

Mitigation and Remediation

To mitigate the potential risk, the project can override the `mint()` function, add a priviledged `burn()` function to clear the attacker's minted `mpETH`. Another optional choice is to deprecating the current `mpETH` token and redeploying a new token/staking contract.

Note that the majority of funds have ultimately been consolidated into the address 0x45dd...910a, which we are actively monitoring for further movements across chains or interactions.
The specific mitigation and remediation plan is still under discussion.