Analysis of Exactly Protocol's $7.3M Exploit: How the Permit Check is Bypassed

Beosin · Follow

4 min read · 3 days ago

On August 18, 2023, According to Beosin EagleEye monitoring, the Exactly Protocol on Optimism was attacked for $7.3 million.

According to Exactly Protocol, they were "trying to communicate with the attackers to return the stolen assets. Police reports have already been filed". On

Aug 20, the protocol was unpaused, users were able to perform all operations, and no liquidations have occurred.

Here’s the analysis of the exploit.

Related Info

● Attack Txs

0x3d6367de5c191204b44b8a5cf975f257472087a9aadc59b5d744ffdef33a520e

0x1526acfb7062090bd5fed1b3821d1691c87f6c4fb294f56b5b921f0edf0cfad6

0xe8999fb57684856d637504f1f0082b69a3f7b34dd4e7597bea376c9466813585

● Attacker’s address

0x3747dbbcb5c07786a4c59883e473a2e38f571af9

● Attack contracts

0x6dd61c69415c8ecab3fefd80d079435ead1a5b4d

0x995a24c99ea2fd6c87421d516216d9bdc7fa72b4

● Victim contracts

0x16748cb753a68329ca2117a7647aa590317ebf41

Vulnerability Analysis

Multiple market address parameters in the vulnerable contract could be manipulated. The attacker passed in a malicious market contract address, bypassing the permit check, and executed a malicious deposit function to steal the user's USDC collateral and liquidate assets, ultimately achieving profit for the attacker.

Attack Flow

Take the tx 0x3d6367...520e as an example.

Main Steps

Bypass permit check to change _msgSender to victim -> Re-enter to steal victim's assets -> Liquidate victim's assets

Attack Preparation Phase:

The attacker created multiple malicious Market contracts.

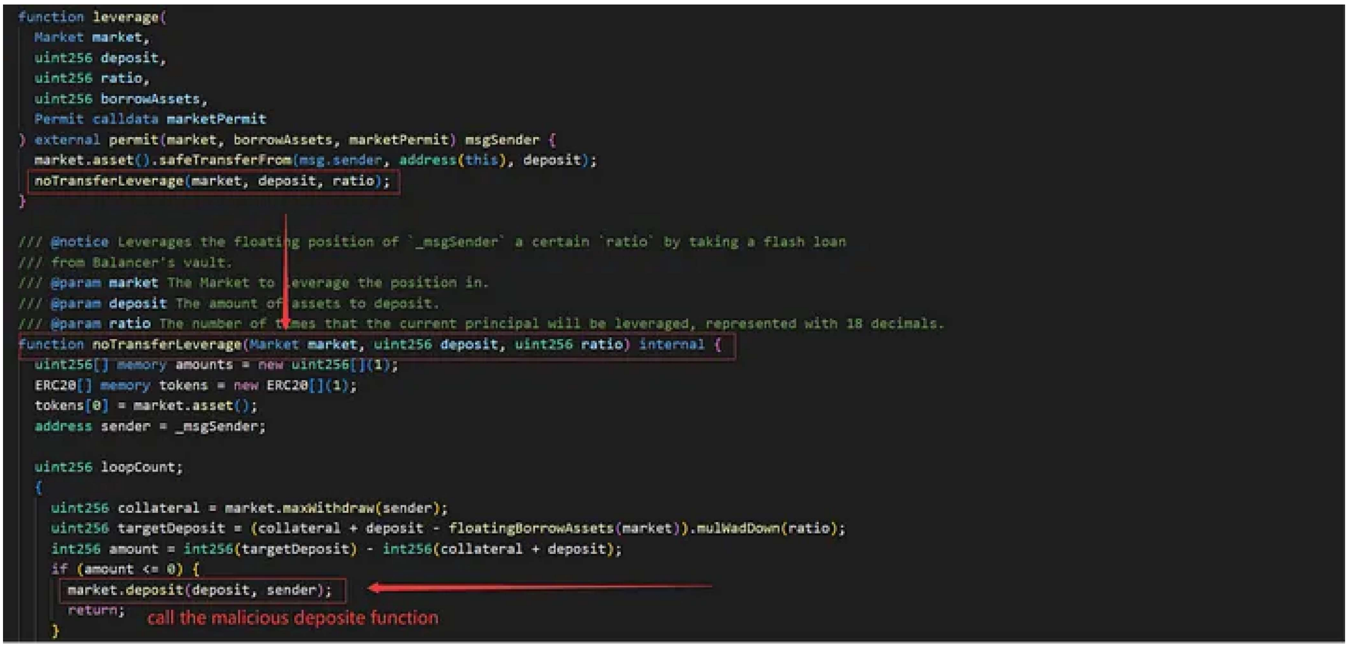
Attack Phase

1. The attacker calls the leverage function of the vulnerable contract and passes in a forged market contract address. Since the market address is not verified, the permit check is bypassed and _msgSender is changed to the victim’s address.


```
function leverage(
    Market market,
    uint256 deposit,
    uint256 ratio,
    uint256 borrowAssets,
    Permit calldata marketPermit
) external permit(market, borrowAssets, marketPermit) msgSender {
    market.asset().safeTransferFrom(msg.sender, address(this), deposit);
    noTransferLeverage(market, deposit, ratio);
}
```

```
modifier permit(
  ERC20 token,
  uint256 assets,
  Permit calldata p
) {
  IERC20PermitUpgradeable(address(token)).safePermit(p.account, address(this), assets, p.deadline, p.v, p.r, p.s);
  {
    address sender = _msgSender;
    if (sender == address(0)) _msgSender = p.account;
    else assert(p.account == sender);
  }
  _;
  assert(_msgSender == address(0));
}
```

2. The leverage function will continue to call the deposit function in the malicious market contract, executing the attacker’s malicious code.

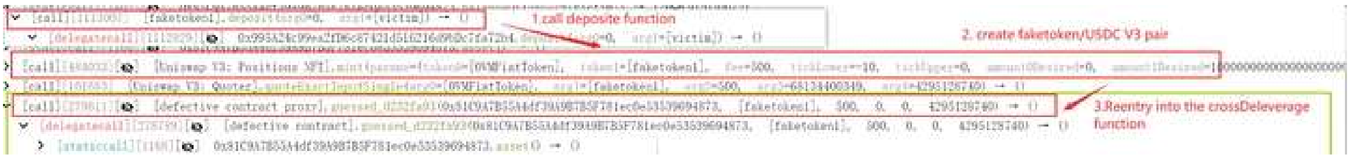


```
function leverage(
  Market market,
  uint256 deposit,
  uint256 ratio,
  uint256 borrowAssets,
  Permit calldata marketPermit
) external permit(market, borrowAssets, marketPermit) msgSender {
  market.asset().safeTransferFrom(msg.sender, address(this), deposit);
  noTransferLeverage(market, deposit, ratio);
}

/// @notice Leverages the floating position of `_msgSender` a certain `ratio` by taking a flash loan
/// from Balancer's vault.
/// @param market The Market to leverage the position in.
/// @param deposit The amount of assets to deposit.
/// @param ratio The number of times that the current principal will be leveraged, represented with 18 decimals.
function noTransferLeverage(Market market, uint256 deposit, uint256 ratio) internal {
  uint256[] memory amounts = new uint256[](1);
  ERC20[] memory tokens = new ERC20[](1);
  tokens[0] = market.asset();
  address sender = _msgSender;

  uint256 loopCount;
  {
    uint256 collateral = market.maxWithdraw(sender);
    uint256 targetDeposit = (collateral + deposit - floatingBorrowAssets(market)).mulWadDown(ratio);
    int256 amount = int256(targetDeposit) - int256(collateral + deposit);
    if (amount <= 0) {
      market.deposit(deposit, sender);
      return;
    }
```


3.The malicious code in the deposit function will first create a malicious V3 token/USDC pool, and then re-enter the crossDeleverage function in the vulnerable contract. Since both marketIn and marketOut are controllable, it results in the V3 pool calculated by the crossDeleverage function becoming the V3 pool created by the attacker.



```
/// @param sqrtPriceLimitX96 The Q64.96 sqrt price limit. If zero for one, the price cannot be less than this.
function crossDeleverage(
  Market marketIn,
  Market marketOut,
  uint24 fee,
  uint256 withdraw,
  uint256 ratio,
  uint160 sqrtPriceLimitX96
) public msgSender {
  LeverageVars memory v;
  v.assetIn = address(marketIn.asset());
  v.assetOut = address(marketOut.asset());
  v.sender = _msgSender;

  v.amount =
    floatingBorrowAssets(marketOut) -
    (
      ratio > 1e18
        ? previewAssetsOut(
          marketIn,
          marketOut,
          (crossPrincipal(marketIn, marketOut, 0, v.sender) - withdraw).mulWadDown(ratio - 1e18)
        )
        : 0
    );

  PoolKey memory poolKey = PoolAddress.getPoolKey(v.assetIn, v.assetOut, fee);
  IUniswapV3Pool(PoolAddress.computeAddress(uniswapV3Factory, poolKey)).swap(
    address(this),
    v.assetIn == poolKey.token0,
    -int256(v.amount),
    sqrtPriceLimitX96,
    abi.encode(
      SwapCallbackData({
        marketIn: marketIn,
        marketOut: marketOut,
        assetIn: v.assetIn,
        assetOut: v.assetOut,
        principal: withdraw,
        account: v.sender,
        fee: fee,
        leverage: false
      })
    )
  );
}
```

faketoken/USDC pair

4.At this point, since _msgSender has been modified to the victim's address, the crossDeleverage function further calls the swap function of the V3 pool created by the attacker as a flash loan, and transfers the victim's funds into the V3 pool in the uniswapV3callback callback function.


1.call deposite function
2. create faketoken/USDC V3 pair
3.Reentry into the crossDeleverage function
4. transfer the victim's collateral(USDC) into the pair in uniswapV3Callback function

```
function crossDeleverage(
  Market marketIn,
  Market marketOut,
  uint24 fee,
  uint256 withdraw,
  uint256 ratio,
  uint160 sqrtPriceLimitX96
) public msgSender {
  LeverageVars memory v;
  v.assetIn = address(marketIn.asset());
  v.assetOut = address(marketOut.asset());
  v.sender = _msgSender;      victim address

  v.amount =
    floatingBorrowAssets(marketOut) -
    (
      ratio > 1e18
        ? previewAssetsOut(
          marketIn,
          marketOut,
          (crossPrincipal(marketIn, marketOut, 0, v.sender) - withdraw).mulWadDown(ratio - 1e18)
        )
        : 0
    );
                                  faketoken/USDC pair
  PoolKey memory poolKey = PoolAddress.getPoolKey(v.assetIn, v.assetOut, fee);
  IUniswapV3Pool(PoolAddress.computeAddress(uniswapV3Factory, poolKey)).swap(
    address(this),
    v.assetIn == poolKey.token0,
    -int256(v.amount),
    sqrtPriceLimitX96,
    abi.encode(
      SwapCallbackData({
        marketIn: marketIn,
        marketOut: marketOut,
        assetIn: v.assetIn,
        assetOut: v.assetOut,
        principal: withdraw,
        account: v.sender,      encode victim address into callBackData
        fee: fee,
        leverage: false
      })
    )
  );
}
```

```
function withdraw(
    uint256 assets,
    address receiver,      v3 pair address
    address owner
) public virtual returns (uint256 shares) {
    shares = previewWithdraw(assets); // No need to check for rounding error, previewWithdraw rounds up.

    if (msg.sender != owner) {
        uint256 allowed = allowance[owner][msg.sender]; // Saves gas for limited approvals.

        if (allowed != type(uint256).max) allowance[owner][msg.sender] = allowed - shares;
    }

    beforeWithdraw(assets, shares);

    _burn(owner, shares);

    emit Withdraw(msg.sender, receiver, owner, assets, shares);

    asset.safeTransfer(receiver, assets);      transfer usdc to fakeToken/USDC pair
}
```

5.The attacker removes liquidity to drain the victim's funds from the V3 pool.

6.Since the victim's deposited funds were transferred away, meeting the liquidation criteria, the attacker further liquidates the victim's position to gain more attack proceeds.

Fund Flow

As of this writing, the stolen funds have been bridged cross-chain to Ethereum via the Optimism bridge and Across Protocol.

Summary

It is recommended that contract addresses used as LP tokens be whitelisted to prevent malicious manipulation. Currently, Beosin has conducted security audits for multiple projects on Optimism such as DIPX, so Beosin recommends that projects undergo comprehensive security audits by professional security audit firms before launch to mitigate security risks.

About Beosin

Beosin is a leading global blockchain security company co-founded by several professors from world-renowned universities and there are 40+ PhDs in the team, and set up offices in 10+ cities including Hong Kong, Singapore, Tokyo and Miami. With the mission of “Securing Blockchain Ecosystem”, Beosin provides “All-in-one” blockchain security solution covering Smart Contract Audit, Risk Monitoring & Alert, KYT/AML, and Crypto Tracing. Beosin has already audited more than 3000 smart contracts including famous Web3 projects PancakeSwap, Uniswap, DAI, OKSwap and all of them are monitored by Beosin EagleEye. The KYT AML are serving 100+ institutions including Binance.

Contact

If you need any blockchain security services, welcome to contact us:

Official Website Beosin EagleEye Twitter Telegram Linkedin