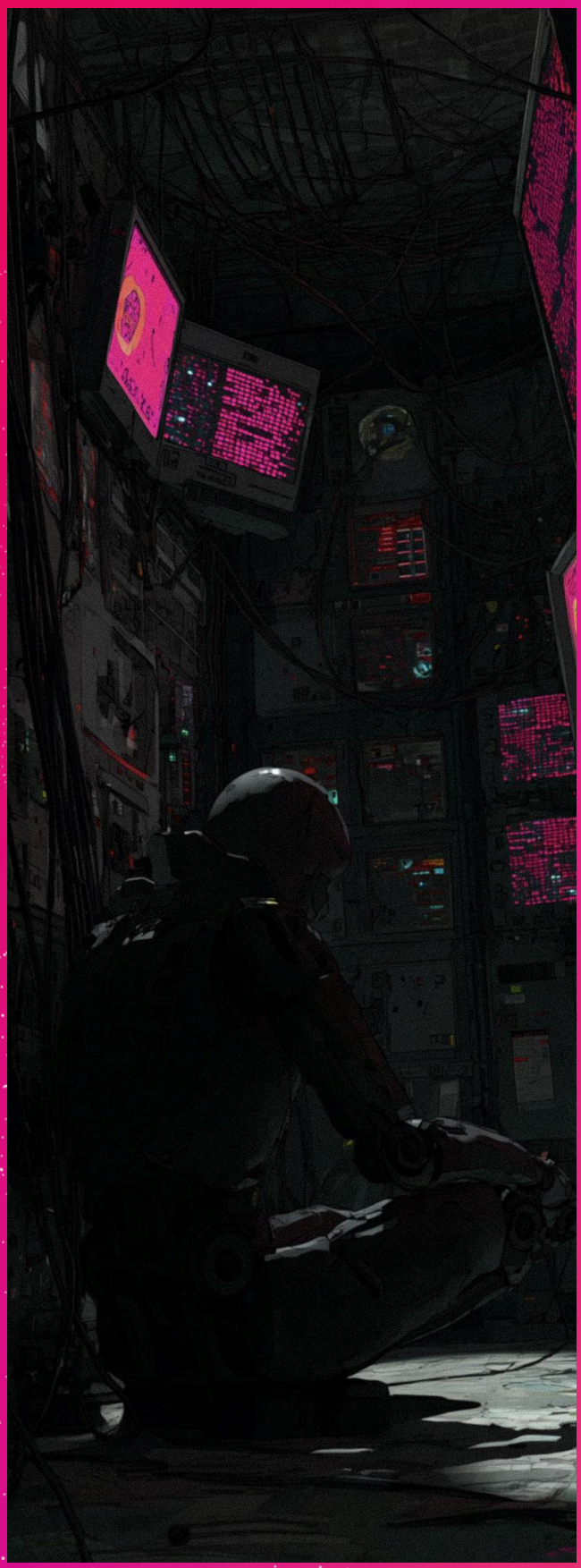
IMMUNEFI AUDIT

Immunefi / Plaza



DATE June 11, 2025

AUDITOR **DrasticWaterMelon** and **Panda**, Security Researchers

REPORT BY **gmhacker**, Immunefi Head of Security

ABOUT IMMUNEFI

Immunefi is the leading onchain security platform, having directly prevented hacks worth more than $25 billion USD. Immunefi security researchers have earned over $120M USD for responsibly disclosing over 4,000 web2 and web3 vulnerabilities, more than the rest of the industry combined.

Through Magnus, Immunefi delivers a comprehensive suite of best-in-class security services through a single command center to more than 300 projects — including Sky (formerly MakerDAO), Optimism, Polygon, GMX, Reserve, Chainlink, TheGraph, Gnosis Chain, Lido, LayerZero, Arbitrum, StarkNet, EigenLayer, AAVE, ZKsync, Morpho, Ethena, USDT0, Stacks, Babylon, Fuel, Sei, Scroll, XION, Wormhole, Firedancer, Jito, Pyth, Eclipse, PancakeSwap and many more.

Magnus unifies SecOps across the entire onchain lifecycle, combining Immunefi's market leading products and community of elite security researchers with a curated set of the very best security products and technologies provided by top security firms — including Runtime Verification, Dedaub, Fuzzland, Nexus Mutual, Failsafe, OtterSec and others.

Magnus is powered by Immunefi's proprietary vulnerabilities dataset — the largest and most comprehensive in web3, ensuring that security leaders and teams have the best possible tools for identifying and mitigating life threats before they cause catastrophic harm, all while reducing operational overhead and complexity.

Learn how you can benefit too at immunefi.com.

TERMINOLOGY

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- **Likelihood** represents the likelihood of a finding to be triggered or exploited in practice
- **Impact** specifies the technical and business-related consequences of a finding
- **Severity** is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

LIKELIHOOD	IMPACT		
	HIGH	MEDIUM	LOW
CRITICAL	Critical	Critical	High
HIGH	High	High	Medium
MEDIUM	Medium	Medium	Low
LOW	Low		
NONE	None		

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

EXECUTIVE SUMMARY

Over the course of 5 days in total, Plaza Finance engaged with Immunefi to review the cross-chain LP distributor contracts. In this period of time a total of 12 issues were identified.

SUMMARY

Name	Plaza Finance
Repository	https://github.com/Convexity-Research/plaza-evm/tree/80e53319a7a4862f823d420e61e40dce36e03808
Type of Project	Defi Derivatives
Audit Timeline	May 16th - May 22nd
Fix Period	June 10th

ISSUES FOUND

Severity	Count	Fixed	Acknowledged
Critical	1	1	0
High	2	2	0
Medium	1	1	0
Low	2	2	0
Insights	6	6	0

CATEGORY BREAKDOWN

Bug	6
Gas Optimization	0
Informational	6

FINDINGS

IMM-CRIT-01

Double-Claim via Cross-Chain Transfers

Id	IMM-CRIT-01
Severity	Critical
Category	Bug
Status	Fixed in `dfea08d31841e63858d129628be000d7cd24be67`

Description

A user can potentially claim rewards on a chain, transfer their tokens to a different chain, and claim again on the remote chain. This is possible if the system does not globally track which tokens have already been claimed for a given period across all chains.

Impact:

- Dishonest users can receive more than their fair share of rewards.
- The system can be drained of USDC.
- Honest users may be left with nothing to claim.

Root Cause:

- Claims are tracked per chain, not globally.
- There is no mechanism to prevent a token/account from claiming on multiple chains for the same period.
- Transactions are not atomic and guaranteed to be executed in order.

Recommendation

Transfer tokens with the period information

By transferring tokens with the period information, when the tokens arrive on the remote chain, make sure the period matches.

Disable transfers between chains during the period increase

By disabling transfers between chains during the USDC distribution period, it will prevent users from moving tokens after claiming.

IMM-HIGH-01

Transfers between remote chains will cause `BondBaseOftAdapter.remoteBalance` to track incorrect data

Id	IMM-HIGH-01
Severity	High
Category	Bug
Status	Fixed in 92ebdf771dad9a7ada8b76242d1c0006f7a5fbca

Description

BondBaseOftAdapter is used to track the amount of `bondETH` bridged to remote chains via the `remoteBalance` (link here) mapping. When a cross chain transfer between Base and a remote chain occurs, such mapping is updated accordingly to correctly track the amount of `bondETH` currently transferred to the remote chain.

When a cross chain transfer between two remote chains occurs, the `remoteBalance` mapping will not be updated despite both chains' total amounts of `bondETH` having changed.

Impact

Having `remoteBalance` deviate from correctly representing the amount of `bondETH` held in remote chains will cause the following issues:

1. During a call to `BondBaseOftAdapter`.increasePeriodForRemote, the `snapshotBalance` mapping is written the value held remoteBalance for a given remote chain id. This will propagate the error to `CrossChainController._getUsdcAmountForChain` which determines the amount of USDC to be sent to a remote chain, ultimately leading to an incorrect amount of USDC being transferred to remote chains.
2. When bridging `bondETH` back to Base, `BondBaseOftAdapter._credit` is executed as part of the incoming LayerZero processing flow. Such method decreases `remoteBalance` by the bridged amount and then mints such amount to the cross chain transfer recipient: if `remoteBalance` tracks a lower amount than the actual amount of `bondETH` bridged to a given remote chain. In this case, it is possible for transfers from said chain to Base to be correctly initiated on the remote chain, but fail because of an overflow when being processed on Base, leading to stuck `bondETH` tokens for users.

Recommendation

Not sure if its possible to frontrun on MIDL network, but as its EVM based network, there is a possibility so I would Modify the swapCollateral to accept a user-provided `minAmountOut` parameter instead of calculating it on-chain.
Two possible fixes for the highlighted issue were identified:

1. Implement a hub-and-spoke architecture via LayerZero peer permissions: allowing Base to interact with every remote chain, but constraining remote chains to interact solely with Base will disallow remote-to-remote transfers, removing the root cause of the issue altogether.
2. Implement additional functionality within `BondRemoteOftAdapter._credit` and/or `BondRemoteOftAdapter._debit` to send a LayerZero cross chain message to Base's `BondBaseOftAdapter` to have it update the `remoteBalance` mapping accordingly.

IMM-HIGH-02

Cross-Chain USDC Distribution Imbalance Issue

Id	IMM-HIGH-02
Severity	High
Category	Bug
Status	Fixed in dfea08d31841e63858d129628be000d7cd24be67

Description

Issue:

The current cross-chain distribution mechanism for USDC and shares is susceptible to imbalances due to the timing of the snapshot and the actual USDC transfer. The process is as follows:

1. **Snapshot**: The base chain takes a snapshot of balances for each remote chain.
2. **USDC Transfer**: USDC is sent to each remote chain based on the snapshot.
3. **Remote Distribution**: Remote distributors receive USDC and are expected to distribute it according to the snapshot.
4. **Shares per Token Update**: The shares per token value is sent to all remote chains.

Problem:

If tokens are moved between chains between the snapshot and the USDC transfer, the actual balances on remote chains may not match the snapshot. This can result in:

Some remote distributors having insufficient USDC to fulfill claims (causing claim failures).
Others having excess USDC (which may be stuck or misallocated).
This is a classic race condition between the snapshot and the actual transfer, leading to cross-chain state divergence and potential denial of service for users on some chains.

Reference:
See RemoteDistributor.sol#L117-L119 for the claim logic that can fail if the USDC balance is insufficient.

Recommendation

Below are several possible mitigations:

- **Implement a Rebalancing Mechanism**
  Introduce a function/mechanism to rebalance USDC across remote distributors after the initial distribution.

- **Lock Transfers During Distribution**
  Consider pausing or restricting token transfers between the snapshot and the completion of USDC distribution to remote chains. This can be done by:
  - Using the `Pausable` mechanism to temporarily pause transfers.
  - Only allowing transfers after all remote chains have confirmed receipt of USDC.

- **Pull USDC from base on claim**
  Instead of transferring USDC to remote chains, pull USDC from the base chain on claim. This way, the USDC is always there when needed.

This is a simpler mechanism to implement and does not require any additional state or mechanisms.
It is not as gas efficient as transferring USDC once and having it there when needed, but it is more secure.

IMM-MED-01

CCIP messages will fail to be sent correctly

Id	IMM-MED-01
Severity	Medium
Category	Bug
Status	Fixed in cfa39f7201db3159b6222f400100f255932e0d13

Description

`CrossChainController.sendUsdcToRemoteDistributors` may be called to send USDC to remote chains via CCIP.

The method uses the `_buildCCIPMessage internal` method to construct the CCIP message to be sent from Base to any EVM remote chain: every message is built using `feeToken: address(0)`, which may be used to signal to CCIP's router contract that the user intends to pay the necessary fees in the chain's native asset. Because `CrossChainController.sendUsdcToRemoteDistributors` fails to send any native ETH when calling `ccipRouter.ccipSend`, sending CCIP messages will always fail.

Proof of Concept

The following PoC may be added to a `test/CrossChainController.t.sol` file and executed via `forge t --mc Audit`:

```
TypeScript
pragma solidity ^0.8.26;

import {Client, IRouterClient} from "../../src/cross-chain/CrossChainController.sol";
import {UUPSUpgradeable} from "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
import "@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol";

import "forge-std/Test.sol";

interface IApprove {
    function approve(address, uint) external;
}

contract Audit is Test {
    address usdc;
```

```
    address router;
    address user;

    function setUp() external {
        vm.createSelectFork(
            getChain(0x2105).rpcUrl, // fork base
            30516567
        );

        usdc = 0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913;
        router = 0x881e3A65B4d4a04dD529061dd0071cf975F58bCD;
        user = makeAddr("user");

        // Give usdc allowance
        deal(usdc, user, 1_000_000);
        vm.prank(user); IApprove(usdc).approve(router, 1_000_000);
    }

    function test_sendCCIP_fails() external {
        uint64 ethID = 5009297550715157269; // ETH ccip destination chain selector

        // Build CCIP message
        Client.EVMTokenAmount[] memory amounts = new Client.EVMTokenAmount[](1);
        amounts[0] = Client.EVMTokenAmount({token: usdc, amount: 1_000_000});

        Client.EVM2AnyMessage memory message = Client.EVM2AnyMessage({
            receiver: abi.encode(user),
            data: bytes(""),
            tokenAmounts: amounts,
            extraArgs: "",
            feeToken: address(0)
        });

        // Fetch fee
        uint256 fee = IRouterClient(router).getFee(
            ethID,
            message
        );
        assertGt(fee, 0, "!fee");

        startHoax(user); // deal + startPrank

        // CCIP msg send fails if no fee is provided
        vm.expectRevert("InsufficientFeeTokenAmount()");
        IRouterClient(router).ccipSend(
            ethID,
            message
        );

        // CCIP msg is successfully executed
        IRouterClient(router).ccipSend{value: fee}(
            ethID,
            message
        );
    }
}
```

Recommendation

Execute `ccipRouter.ccipSend{value: fees}(..)` in order to forward the correct amount of ETH to cover fees.

IMM-LOW-01

`DistributorAdapter._isPeriodFinalized` considers future periods as finalized

Id	IMM-LOW-01
Severity	LOW
Category	Bug
Status	Fixed in cfa39f7201db3159b6222f400100f255932e0d13

Description

`DistributorAdapter._isPeriodFinalized` is used by the contract to determine whether a given period has been finalized or not, and thus whether rewards may be claimed for such period.

The method's current implementation only verifies that `period == lastPeriod && Auction(pool.auctions(lastPeriod)).state() == Auction.State.BIDDING)` evaluates to `false`. Hence, for any `period` such that `period != lastPeriod, where lastPeriod = _currentPeriod() - 1`, the condition will evaluate to `false` and the method will be considered as finalized.

Recommendation

Modify the DistributorAdapter.sol#L42 condition to check `period >= lastPeriod`, allowing only periods strictly smaller than `lastPeriod`.

IMM-LOW-02

RemoteRoleController uses non-upgradeable dependencies

Id	IMM-LOW-02
Severity	Low
Category	Bug
Status	Fixed in cfa39f7201db3159b6222f400100f255932e0d13

Description

RemoteRoleController inherits OpenZeppelin's UUPSUpgradeable contract, making it able to hold the logic for upgrading an ERC-1967 proxy when it is used as its implementation.

Because this method is intended to be used as the implementation for a proxy contract, it should utilize the upgradeable versions of OpenZeppelin's library contracts, which use custom storage locations in order to ensure that future implementation upgrades do not change the implementation's storage layout, which would lead to corrupting the proxy's storage.

Currently, RemoteRoleController inherits the non-upgradeable AccessControl and Pausable contracts.

Recommendation

Use AccessControlUpgradeable and PausableUpgradeable instead of AccessControl and Pausable.

IMM-INSIGHT-01

Unused imports

Id	IMM-INSIGHT-01
Severity	INSIGHT
Category	Informational
Status	Fixed in cfa39f7201db3159b6222f400100f255932e0d13

Description

The following imports aren't used in the respective contracts:

TypeScript

File: src/cross-chain/BondBaseOftAdapter.sol

```
13:                     import                     {EnforcedOptionParam}                  from
"@layerzerolabs/oapp-evm/contracts/oapp/libs/OAppOptionsType3.sol";

14:                       import                       {IOAppCore}                      from
"@layerzerolabs/oapp-evm/contracts/oapp/interfaces/IOAppCore.sol";
```

src/cross-chain/BondBaseOftAdapter.sol#L13, src/cross-chain/BondBaseOftAdapter.sol#L14

TypeScript

File: src/cross-chain/BondRemoteOftAdapter.sol

```
10: import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";

12:                     import                     {OFTComposeMsgCodec}                   from
"@layerzerolabs/oft-evm/contracts/libs/OFTComposeMsgCodec.sol";
```

src/cross-chain/BondRemoteOftAdapter.sol#L10, src/cross-chain/BondRemoteOftAdapter.sol#L12

TypeScript

File: src/cross-chain/CrossChainController.sol

```
13: import {EnforcedOptionParam} from
"@layerzerolabs/oapp-evm/contracts/oapp/libs/OAppOptionsType3.sol";
```

src/cross-chain/CrossChainController.sol#L13

TypeScript

File: src/cross-chain/RemoteDistributor.sol

```
16: import {IOAppCore} from
"@layerzerolabs/oapp-evm/contracts/oapp/interfaces/IOAppCore.sol";

20: import {IRouterClient} from
"@chainlink/contracts/src/v0.8/ccip/interfaces/IRouterClient.sol";
```

src/cross-chain/RemoteDistributor.sol#L16, src/cross-chain/RemoteDistributor.sol#L20

Recommendation

Remove unused imports.

IMM-INSIGHT-02

DistributorAdapter doesn't disable initializers

Id	IMM-INSIGHT-02
Severity	INSIGHT
Category	Informational
Status	Fixed in cfa39f7201db3159b6222f400100f255932e0d13

Description

DistributorAdapter does not disable initializers during deployment, leaving the contract's implementation open for initialization.

Recommendation

Add `_disableInitializers()` to the contract's constructor.

IMM-INSIGHT-03

`DistributorAdapterCore.claim` should use `MerkleProof.verifyCalldata`

Id	IMM-INSIGHT-03
Severity	INSIGHT
Category	Informational
Status	Fixed in cfa39f7201db3159b6222f400100f255932e0d13

Description

OpenZeppelin's MerkleProof library offers methods to validate a merkle proof located in memory or in calldata:

- MerkleProof.verify
- MerkleProof.verifyCalldata

Recommendation

DistributorAdapterCore.claim may use the calldata version to avoid copying the proof from `calldata` to memory, reducing the method's gas consumption.

IMM-INSIGHT-04

Unused `modifier` definition

Id	IMM-INSIGHT-04
Severity	INSIGHT
Category	Informational
Status	Fixed in cfa39f7201db3159b6222f400100f255932e0d13

Description

The following modifier are never used, consider removing them.

TypeScript
File: src/cross-chain/BondBaseOftAdapter.sol

```
127: modifier onlyPoolFactory() {
128:         if (msg.sender != address(BondToken(address(innerToken)).poolFactory())) revert
OnlyPoolFactory();
129:     _;
130:   }
```

BondBaseOftAdapter.sol#L127

Recommendation

Remove the unused modifier and the error

IMM-INSIGHT-05

Pausable contracts not using modifiers

Id	IMM-INSIGHT-05
Severity	INSIGHT
Category	Informational
Status	Fixed in cfa39f7201db3159b6222f400100f255932e0d13

Description

These contracts import a `pausable` library, but there isn't any use of a pausable modifier.

```
TypeScript
File: src/cross-chain/BondBaseOftAdapter.sol

19: contract BondBaseOftAdapter is
20:   Initializable,
21:   UUPSUpgradeable,
22:   OwnableUpgradeable,
23:   PausableUpgradeable
```

BondBaseOftAdapter.sol#L19

```
TypeScript
File: src/cross-chain/CrossChainController.sol

17: contract CrossChainController is Initializable, UUPSUpgradeable, PausableUpgradeable {
```

CrossChainController.sol#L17

```
TypeScript
File: src/cross-chain/LeverageBaseOftAdapter.sol
```

```
12: contract LeverageBaseOftAdapter is
13:   Initializable,
14:   UUPSUpgradeable,
15:   OwnableUpgradeable,
16:   PausableUpgradeable,
```

LeverageBaseOftAdapter.sol#L12

```
TypeScript
File: src/cross-chain/RemoteRoleController.sol

12: contract RemoteRoleController is Initializable, AccessControl, Pausable, UUPSUpgradeable
{
```

RemoteRoleController.sol#L12

Recommendation

Remove the unused contract.

IMM-INSIGHT-06

Pool.sol imports `CrossChainController` twice

Id	IMM-INSIGHT-06
Severity	INSIGHT
Category	Informational
Status	Fixed in cfa39f7201db3159b6222f400100f255932e0d13

Description

Pool.sol#L15-L16 imports the same contract twice, increasing the contract size and thus its deployment cost.

Recommendation

Avoid importing the same contract twice.