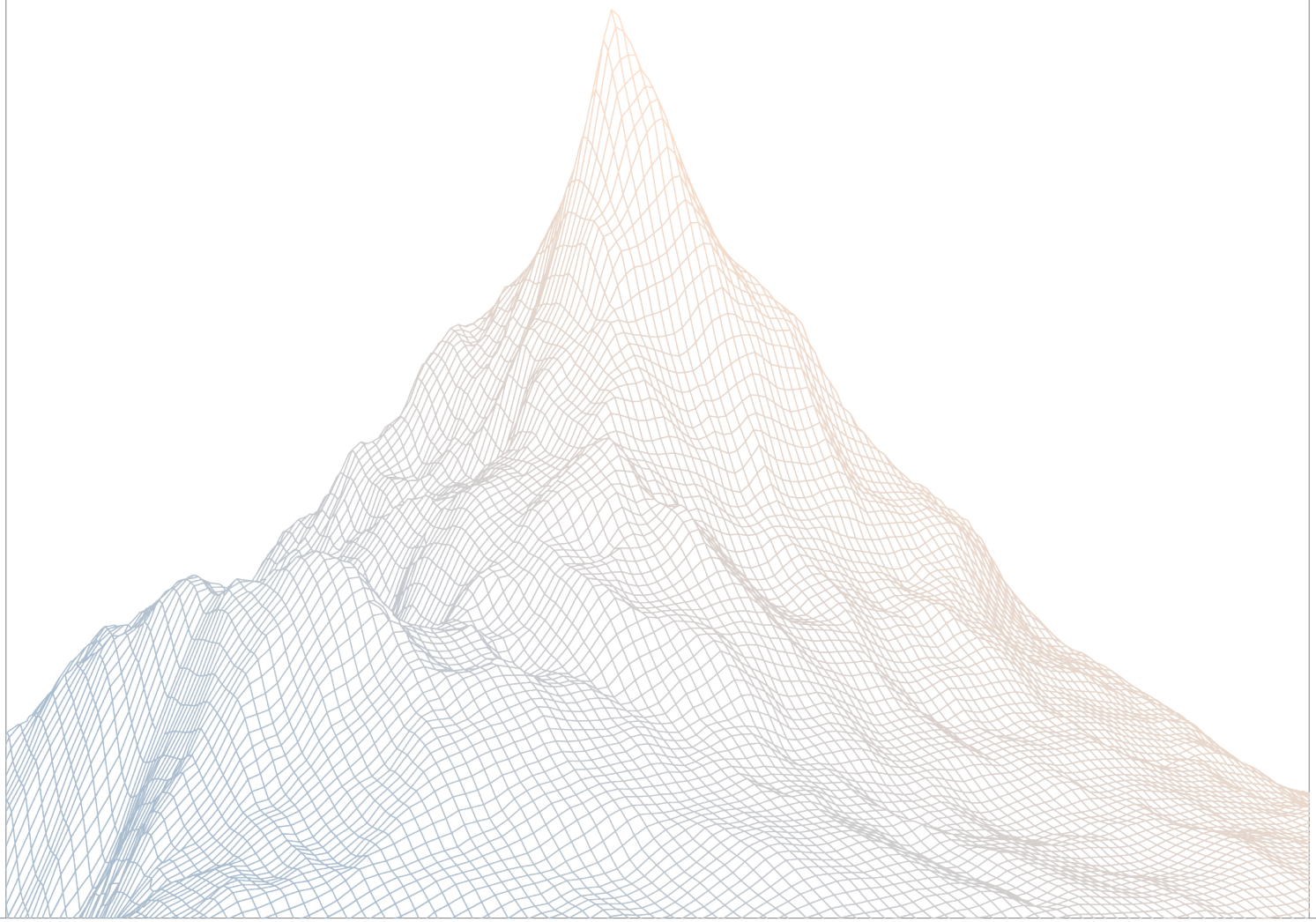
Zenith

PartyDAO

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: February 25th to February 26th, 2025

AUDITED BY: ethersky
ladboy

Contents

1 Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About PartyDAO

The Party Protocol provides on-chain functionality for group formation, coordination, and distribution, with the goal of making Ethereum multiplayer.

The Party Protocol powers secure commercial transactions through crowdfunding, on-chain payments and transactions, and distributions to Party members.

2.2 Scope

The engagement involved a review of the following targets:

Target	yield
Repository	https://github.com/PartyDAO/yield
Commit Hash	`400a9d9ad6dd12958ac4467d6d8278ba56ab6dfd`
Files	`ERC4626CrossChainPort.sol` `ERC4626CrossChainAdapter.sol`

2.3 Audit Timeline

February 25, 2025	Audit start
February 26, 2025	Audit end
March 17, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	8
Medium Risk	4
Low Risk	4
Informational	4
Total Issues	**20**

3

Findings Summary

ID	Description	Status
H-1	The deposit will be reverted in an Adapter because the confirmation relayer fee cannot be sent to ACROSS	Resolved
H-2	The confirmation relayer fee is accidentally removed from the withdrawal amount in an Adapter	Resolved
H-3	The confirmation relayer fee is not considered when depositing assets into an ERC4626 vault in an Adapter	Resolved
H-4	There is a conflict in the encoding and decoding of a message between the Port and Adapter	Resolved
H-5	Arbitrary relayers can be set when submitting either a deposit or a withdrawal batch	Resolved
H-6	The withdrawBatchId is incorrectly updated in the submitUnlockedDepositsWithdrawBatch function	Resolved
H-7	Lack of token allowance when deposit to the vault in Adapter.sol	Resolved
H-8	Refunded deposit is locked in the contract	Resolved
M-1	Users can lose their shares without claiming the actual assets in the claimUnlockedDepositWithdraw function	Resolved
M-2	Revert condition in external call Vault#deposit and Vault#redeem is not handled	Resolved
M-3	Minimum relayer fee should not be proportional to total deposit amount in the Port.sol	Resolved
M-4	User can set confirmation relayer fee or submission relayer fee to 0 to disincentivize relayer from filling the cross-chain message.	Resolved
L-1	There is no function to update approved relayers in the Port	Resolved
L-2	User can frontrun the submission by canceling withdraw to force submitQueuedWithdrawBatch to revert	Acknowledged

ID	Description	Status
L-3	Consider add setter for immutable variable	Resolved
L-4	Consider use uint256 instead of uint16 for round tracking	Resolved
I-1	Users may lose some assets due to the fee	Resolved
I-2	The check for sharesRedeemed can be added to the validateWithdraw function	Resolved
I-3	The 1-hour deadline should be adjusted	Resolved
I-4	The addresses for the contracts on the destination chain can be stored	Acknowledged

4

Findings

4.1 High Risk

A total of 8 high risk findings were identified.

[H-1] The deposit will be reverted in an Adapter because the confirmation relayer fee cannot be sent to ACROSS

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: High

Target

- ERC4626CrossChainAdapter.sol

Description:

When a `deposit` is submitted to the `Adapter`, the minted `shares` amount is confirmed by the `confirmation relayer`. To do this, the `caller` must set the `confirmation relayer fee` when submitting a `deposit` in the `Port`.

However, in the `_onDepositSubmitted` function of the `Adapter`, the `confirmation` is sent using the `_sendMessageAcrossChain` function.

- ERC4626CrossChainAdapter.sol#L100

```
function _onDepositSubmitted(
    uint16 depositRoundId,
    uint96 totalAssetsToDeposit,
    address relayer,
    uint96 relayFee
) internal {
    // Deposit assets into vault and get shares minted
    uint256 sharesMintedFromVault = VAULT.deposit({
        assets: totalAssetsToDeposit,
        receiver: address(this)
    });

    // Send confirmation message back to port
    _sendMessageAcrossChain({
        relayer: relayer,
```

```
        relayFee: relayFee,
        deadline: uint32(block.timestamp + 1 hours),
        message: abi.encode(
            ConfirmationType.Deposit,
            DepositConfirmation({
                confirmationType: ConfirmationType.Deposit,
                depositRoundId: depositRoundId,
                totalSharesMinted: uint96(sharesMintedFromVault)
            })
        )
    });
}
```

This function lacks an `approval` step for `ACROSS`, even though this transaction attempts to send the `confirmation relayer fee` to `ACROSS`.

- ERC4626CrossChainAdapter.sol#L146-L148

```
function _sendMessageAcrossChain(address relayer, uint96 relayFee,
    uint32 deadline, bytes memory message) internal {
    ACROSS.depositV3({
        depositor: address(this),
        recipient: address(PORT),
        inputToken: address(ASSET),
        outputToken: address(PORT_CHAIN_ASSET),
@->         inputAmount: relayFee,  // non-zero
        outputAmount: 0,
        destinationChainId: PORT_CHAIN_ID,
        exclusiveRelayer: relayer,
        quoteTimestamp: uint32(block.timestamp),
        fillDeadline: deadline,
        exclusivityDeadline: deadline,
        message: message
    });
}
```

Since `ACROSS` does not have `approval`, the transaction will be reverted.

Recommendations:

```
   function _onDepositSubmitted(uint16 depositRoundId,
    uint96 totalAssetsToDeposit, address relayer, uint96 relayFee)
    internal {
// Deposit assets into vault and get shares minted
```

```
uint256 sharesMintedFromVault = VAULT.deposit({ assets:
    totalAssetsToDeposit, receiver: address(this) });

// Send confirmation message back to port
   -^^I_sendMessageAcrossChain({ relayer: relayer, relayFee: relayFee,
    deadline: uint32(block.timestamp + 1 hours), message:
    abi.encode(ConfirmationType.Deposit, DepositConfirmation({
    confirmationType: ConfirmationType.Deposit, depositRoundId:
    depositRoundId, totalSharesMinted: uint96(sharesMintedFromVault) })) });
_transferAssetsAcrossChain({
        assets: relayFee,
        relayer: relayer,
        relayFee: relayFee,
        deadline: uint32(block.timestamp + 1 hours),
        message: abi.encode(
            ConfirmationType.Deposit,
            DepositConfirmation({
                confirmationType: ConfirmationType.Deposit,
                depositRoundId: depositRoundId,
            totalSharesMinted: uint96(sharesMintedFromVault)
        })
    )
   });
}
```

PartyDAO: Resolved with PR-9

Zenith: Verified.

[H-2] The confirmation relayer fee is accidentally removed from the withdrawal amount in an Adapter

SEVERITY: High	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: High

Target

- ERC4626CrossChainAdapter.sol

Description:

Suppose the `caller` sets `100` as the `submission relayer fee` and `100` as the `confirmation relayer fee`, and submits a `withdrawal batch` using the `submitQueuedWithdrawBatch` function.

- ERC4626CrossChainPort.sol#L344-L348

```
function submitQueuedWithdrawBatch(
    address[] calldata users,
    address submissionRelayer,
    uint96 submissionRelayFee,
    address confirmationRelayer,
    uint96 confirmationRelayFee,
    Permit2 memory permit
) external payable returns (uint16 withdrawBatchId) {
    WithdrawBatch storage withdrawBatch = withdrawBatches[withdrawBatchId];
    withdrawBatch.totalSharesRedeemed = totalSharesToRedeem;
    withdrawBatch.totalAssetsDeposited = totalAssetsDeposited;

    _transferAssetsAcrossChain({
@->         assets: totalRelayFee, // 200
        relayer: submissionRelayer,
@->         relayFee: submissionRelayFee, // 100
        deadline: uint32(block.timestamp + 1 hours),
        message: abi.encode(
            WithdrawSubmission({
                submissionType: SubmissionType.Withdraw,
                withdrawBatchId: withdrawBatchId,
                totalSharesToRedeem: totalSharesToRedeem,
                confirmationRelayer: confirmationRelayer,
```

```
                confirmationRelayFee: confirmationRelayFee
            })
        )
    });
}
```

The total amount of `200` (which is the sum of both `fees`) is transferred to `ACROSS`, and the `confirmation relayer fee` is transferred to the `Adapter` for `confirmation`.

In the `_onWithdrawSubmitted` function, the `assets` related to the redeemed `shares` are withdrawn from the `vault` at `line 116`.

- ERC4626CrossChainAdapter.sol#L116

```
function _onWithdrawSubmitted(
    uint16 withdrawBatchId,
    uint96 totalSharesToRedeem,
    address relayer,
    uint96 relayFee
) internal {
116:  uint256 assetsWithdrawnFromVault = VAULT.redeem({
        shares: totalSharesToRedeem,
        receiver: address(this),
        owner: address(this)
    });

    _transferAssetsAcrossChain({
        assets: uint96(assetsWithdrawnFromVault),
        relayer: relayer,
@->         relayFee: relayFee, // 100
        deadline: uint32(block.timestamp + 1 hours),
        message: abi.encode(
            ConfirmationType.Withdraw,
            WithdrawConfirmation({
                confirmationType: ConfirmationType.Withdraw,
                withdrawBatchId: withdrawBatchId
            })
        )
    });
}
```

The sum of these `withdrawn assets` and the `confirmation relayer fee` (`100`) should be transferred to `ACROSS`. By doing this, the `confirmation relayer fee` will be charged by the `relayer` on the `ACROSS` side, and the exact `withdrawal amount` will be transferred to the `Port`.

However, only the `withdrawn assets` are transferred to ACROSS, and unfortunately, the defined `confirmation relayer fee` is deducted from this amount.

Recommendations:

```
function _onWithdrawSubmitted(
    uint16 withdrawBatchId,
    uint96 totalSharesToRedeem,
    address relayer,
    uint96 relayFee
) internal {
    // Redeem shares from vault for assets
    uint256 assetsWithdrawnFromVault = VAULT.redeem({
        shares: totalSharesToRedeem,
        receiver: address(this),
        owner: address(this)
    });

    // Transfer assets back to port with confirmation
    _transferAssetsAcrossChain({
        assets: uint96(assetsWithdrawnFromVault),
        assets: uint96(assetsWithdrawnFromVault + relayFee),
        relayer: relayer,
        relayFee: relayFee,
        deadline: uint32(block.timestamp + 1 hours),
        message: abi.encode(
            ConfirmationType.Withdraw,
            WithdrawConfirmation({
                confirmationType: ConfirmationType.Withdraw,
                withdrawBatchId: withdrawBatchId
            })
        )
    });
}
```

PartyDAO: Resolved with PR-9

Zenith: Verified

[H-3] The confirmation relayer fee is not considered when depositing assets into an ERC4626 vault in an Adapter

SEVERITY: High	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: High

Target

- ERC4626CrossChainAdapter.sol

Description:

Suppose the total amount in current `deposit round` is `2000`, and this amount is submitted. The caller sets `100` as the `submission relayer fee` and `100` as the `confirmation relayer fee`. Therefore, the total assets to be transferred to ACROSS are `2200`.

- ERC4626CrossChainPort.sol#L232

```
function submitDepositRound(
    address submissionRelayer,
    uint96 submissionRelayFee,
    address confirmationRelayer,
    uint96 confirmationRelayFee,
    Permit2 memory permit
) external returns (uint16 depositRoundId) {
@->     uint96 totalRelayFee = submissionRelayFee + confirmationRelayFee;  //
    200

    _transferAssetsAcrossChain({
@->         assets: depositRound.totalAssetsDeposited + totalRelayFee,  //
    2200
        relayer: submissionRelayer,
@->         relayFee: submissionRelayFee, // 100
        deadline: uint32(block.timestamp + 1 hours),
        message: abi.encode(
            DepositSubmission({
                submissionType: SubmissionType.Deposit,
                depositRoundId: depositRoundId,
                confirmationRelayer: confirmationRelayer,
                confirmationRelayFee: confirmationRelayFee
            })
```

```
        )
    });

    _incrementDepositRound();
}
```

When the `depositV3` function of `ACROSS` is called, the `input amount` is `2200`, and the `output amount` is `2100`.

- ERC4626CrossChainPort.sol#L488

```
function _transferAssetsAcrossChain(
    uint96 assets,
    address relayer,
    uint96 relayFee,
    uint32 deadline,
    bytes memory message
) internal {
    ASSET.approve({ spender: address(ACROSS), value: assets });
    ACROSS.depositV3({
        depositor: address(this),
        recipient: address(ADAPTER),
        inputToken: address(ASSET),
        outputToken: address(ADAPTER_CHAIN_ASSET),
@->         inputAmount: assets, // 2200
@->         outputAmount: assets - relayFee, // 2200 - 100 = 2100
        destinationChainId: ADAPTER_CHAIN_ID,
        exclusiveRelayer: relayer,
        quoteTimestamp: uint32(block.timestamp),
        fillDeadline: deadline,
        exclusivityDeadline: deadline,
        message: message
    });
}
```

The difference `100` is considered the `relay fee` in `ACROSS` (specifically for the `submission relayer`). `ACROSS` will transfer `2100 tokens` to the `Adapter` on the destination chain.

In the `_onDepositSubmitted` function, the `totalAssetsToDeposit` parameter is `2100`.

- ERC4626CrossChainAdapter.sol#L97

```
function _onDepositSubmitted(
    uint16 depositRoundId,
    uint96 totalAssetsToDeposit,
    address relayer,
```

```
    uint96 relayFee
) internal {
    // Deposit assets into vault and get shares minted
    uint256 sharesMintedFromVault = VAULT.deposit({
@->         assets: totalAssetsToDeposit, // 2100
        receiver: address(this)
    });

    // Send confirmation message back to port
    _sendMessageAcrossChain({
        relayer: relayer,
@->         relayFee: relayFee, // 100
        deadline: uint32(block.timestamp + 1 hours),
        message: abi.encode(
            ConfirmationType.Deposit,
            DepositConfirmation({
                confirmationType: ConfirmationType.Deposit,
                depositRoundId: depositRoundId,
                totalSharesMinted: uint96(sharesMintedFromVault)
            })
        )
    });
}
```

However, these `2100` tokens are `deposited` to the `vault`, which also includes the `confirmation relayer fee` that will be used to send `confirmation` back to the `Port`.

Recommendations:

```
function _onDepositSubmitted(
    uint16 depositRoundId,
    uint96 totalAssetsToDeposit,
    address relayer,
    uint96 relayFee
) internal {
    // Deposit assets into vault and get shares minted
    uint256 sharesMintedFromVault = VAULT.deposit({
        assets: totalAssetsToDeposit,
        assets: totalAssetsToDeposit - relayFee,
        receiver: address(this)
    });

    // Send confirmation message back to port
    _sendMessageAcrossChain({
```

```
            relayer: relayer,
            relayFee: relayFee,
            deadline: uint32(block.timestamp + 1 hours),
            message: abi.encode(
                ConfirmationType.Deposit,
                DepositConfirmation({
                    confirmationType: ConfirmationType.Deposit,
                    depositRoundId: depositRoundId,
                    totalSharesMinted: uint96(sharesMintedFromVault)
                })
            )
        });
    }
```

PartyDAO: Approve Across to spend relayFee by calling SafeERC20.forceApprove(). Patched indirectly in and PR-9 and PR-11

Zenith: Verified

[H-4] There is a conflict in the encoding and decoding of a message between the Port and Adapter

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: High

Target

- ERC4626CrossChainPort.sol
- ERC4626CrossChainAdapter.sol

Description:

When submitting a `deposit` from a `Port` to an `Adapter`, the `message` is simply an encoding of the `DepositSubmission` struct.

- ERC4626CrossChainPort.sol#L232

```
function submitDepositRound(
    address submissionRelayer,
    uint96 submissionRelayFee,
    address confirmationRelayer,
    uint96 confirmationRelayFee,
    Permit2 memory permit
) external returns (uint16 depositRoundId) {
    _transferAssetsAcrossChain({
        assets: depositRound.totalAssetsDeposited + totalRelayFee,
        relayer: submissionRelayer,
        relayFee: submissionRelayFee,
        deadline: uint32(block.timestamp + 1 hours),
        message: abi.encode(
            DepositSubmission({
                submissionType: SubmissionType.Deposit,
                depositRoundId: depositRoundId,
                confirmationRelayer: confirmationRelayer,
                confirmationRelayFee: confirmationRelayFee
            })
        )
    });
}
```

However, in the `Adapter`, it is decoded into a combination of `ERC4626CrossChainPort.SubmissionType` and `DepositSubmission` struct.

- ERC4626CrossChainAdapter.sol#L169

```
function handleV3AcrossMessage(
    address tokenSent,
    uint256 amount,
    address relayer,
    bytes memory message
) external onlyAcross {
    require(isApprovedRelayer[relayer], "Unauthorized relayer");

    (ERC4626CrossChainPort.SubmissionType submissionType) =
        abi.decode(message, (ERC4626CrossChainPort.SubmissionType));

    if (submissionType == ERC4626CrossChainPort.SubmissionType.Deposit) {
        require(tokenSent == address(ASSET), "Unexpected token received");
        // Handle deposit submission
        (, ERC4626CrossChainPort.DepositSubmission memory depositSubmission)
    = abi.decode(
            message,
@->              (ERC4626CrossChainPort.SubmissionType,
    ERC4626CrossChainPort.DepositSubmission)
        );
        _onDepositSubmitted({
            depositRoundId: depositSubmission.depositRoundId,
            totalAssetsToDeposit: uint96(amount),
            relayer: depositSubmission.confirmationRelayer,
            relayFee: depositSubmission.confirmationRelayFee
        });
    }
}
```

The same issue occurs for `withdrawals`.

In contrast, for confirmations from an `Adapter` to a `Port`, the `message` is an encoding of `ConfirmationType.Deposit` and `DepositConfirmation`.

- ERC4626CrossChainAdapter.sol#L100

```
function _onDepositSubmitted(
    uint16 depositRoundId,
    uint96 totalAssetsToDeposit,
    address relayer,
    uint96 relayFee
```

```
) internal {
    _sendMessageAcrossChain({
        relayer: relayer,
        relayFee: relayFee,
        deadline: uint32(block.timestamp + 1 hours),
        message: abi.encode(
            ConfirmationType.Deposit,
            DepositConfirmation({
                confirmationType: ConfirmationType.Deposit,
                depositRoundId: depositRoundId,
                totalSharesMinted: uint96(sharesMintedFromVault)
            })
        )
    });
}
```

However, in the `Port`, it is decoded to `DepositConfirmation` only.

- ERC4626CrossChainPort.sol#L473

```
function handleV3AcrossMessage(
    address tokenSent,
    uint256 amount,
    address relayer,
    bytes memory message
) external onlyAcross {
    require(isApprovedRelayer[relayer], "Unauthorized relayer");

    (
        ERC4626CrossChainAdapter.ConfirmationType confirmationType
    ) = abi.decode(message, (ERC4626CrossChainAdapter.ConfirmationType));
    if (confirmationType ==
    ERC4626CrossChainAdapter.ConfirmationType.Deposit) {
        (
            ERC4626CrossChainAdapter.DepositConfirmation memory confirmation
@->        ) = abi.decode(message,
    (ERC4626CrossChainAdapter.DepositConfirmation));
        _onDepositConfirmed(confirmation.depositRoundId,
    confirmation.totalSharesMinted);
    }
}
```

This conflict will block all communication between a `Port` and an `Adapter`.

Recommendations:

Update the encoding and decoding process between a `Port` and an `Adapter` and add thorough testing to ensure the cross-chain message works as intended.

PartyDAO: Ensure encoding/decoding logic consistency between contracts. Don't separately decode the SubmissionType, don't separately encode the ConfirmationType. Patched indirectly by removing need for SubmissionType and ConfirmationType in PR-9

Zenith: Verified

[H-5] Arbitrary relayers can be set when submitting either a deposit or a withdrawal batch

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- ERC4626CrossChainPort.sol

Description:

When submitting a `deposit`, the caller sets two `relayers`: the `submission relayer` and the `confirmation relayer`. However, there is no check to verify if these `relayers` are approved.

- ERC4626CrossChainPort.sol#L232

```
function submitDepositRound(address submissionRelayer,
    uint96 submissionRelayFee, address confirmationRelayer,
    uint96 confirmationRelayFee, Permit2 memory permit)
    external returns (uint16 depositRoundId) {
    depositRoundId = latestDepositRoundId;
    DepositRound storage depositRound = depositRounds[depositRoundId];

    require(depositRound.submittedAt == 0, "Deposit round already
    submitted");
    require(depositRound.totalAssetsDeposited > 0, "No assets to submit");

    // Bridge assets and send submission message to adapter
    _transferAssetsAcrossChain({ assets: depositRound.totalAssetsDeposited
    + totalRelayFee, relayer: submissionRelayer, relayFee:
    submissionRelayFee, deadline: uint32(block.timestamp + 1 hours),
    message: abi.encode(DepositSubmission({ submissionType:
    SubmissionType.Deposit, depositRoundId: depositRoundId,
    confirmationRelayer: confirmationRelayer, confirmationRelayFee:
    confirmationRelayFee })) });

    _incrementDepositRound();
}
```

Additionally, the `fillDeadline` and `exclusivityDeadline` are both set to `block.timestamp`

`+ 1 hour.`

- ERC4626CrossChainPort.sol#L488

```
function _transferAssetsAcrossChain(uint96 assets, address relayer,
    uint96 relayFee, uint32 deadline, bytes memory message) internal {
    ASSET.approve({ spender: address(ACROSS), value: assets });
    ACROSS.depositV3({ depositor: address(this), recipient:
    address(ADAPTER), inputToken: address(ASSET), outputToken:
    address(ADAPTER_CHAIN_ASSET), inputAmount: assets, outputAmount: assets
    - relayFee, destinationChainId: ADAPTER_CHAIN_ID, exclusiveRelayer:
    relayer, quoteTimestamp: uint32(block.timestamp), fillDeadline:
    deadline, exclusivityDeadline: deadline, message: message });
}
```

This means that only the specified `relayer` can execute the transaction within `ACROSS`.

The `exclusivityDeadline` allows only the specified `relayer` to call the `fillRelay` function within one hour.

- SpokePool.sol#L967-L972

```
function fillRelay(
    V3RelayData memory relayData,
    uint256 repaymentChainId,
    bytes32 repaymentAddress
) public override nonReentrant unpausedFills {
    // Exclusivity deadline is inclusive and is the latest timestamp that the
    exclusive relayer has sole right
    // to fill the relay.
    if (
        _fillIsExclusive(relayData.exclusivityDeadline,
    uint32(getCurrentTime())) &&
        relayData.exclusiveRelayer.toAddress() != msg.sender
    ) {
        revert NotExclusiveRelayer();
    }

    _fillRelayV3(relayExecution, repaymentAddress, false);
}
```

After one hour, anyone can call this function, but since this exceeds the `fillDeadline`, the transaction will be reverted on the destination chain.

- SpokePool.sol#L1648

```
function _fillRelayV3(
    V3RelayExecutionParams memory relayExecution,
    bytes32 relayer,
    bool isSlowFill
) internal {
    V3RelayData memory relayData = relayExecution.relay;

    if (relayData.fillDeadline < getCurrentTime())
    revert ExpiredFillDeadline();

}
```

As a result, setting arbitrary `relayers` effectively blocks the submission of a `deposit` and `withdrawal batch`.

Recommendations:

Add a check to ensure that the two `relayers` are approved in the `submitDepositRound`, `submitQueuedWithdrawBatch`, and `submitUnlockedDepositsWithdrawBatch` functions

PartyDAO: Add validation to ensure only approved relayers are used for cross-chain messages. Add setApprovalForRelayers() function to the Port contract similar to the one in the Adapter. Patched in PR-19

Zenith: Verified

[H-6] The withdrawBatchId is incorrectly updated in the submitUnlockedDepositsWithdrawBatch function

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- ERC4626CrossChainPort.sol

Description:

Suppose the `latestWithdrawBatchId` is currently 2. When a `batch` of `queued withdrawals` is submitted for some users using the `submitQueuedWithdrawBatch` function:

- At `line 322`, `latestWithdrawBatchId` increases by 1, and `withdrawBatchId` is set to this new value, which is 3.
- At `line 345`, the shares to be redeemed are stored in this `withdrawal batch`.
- ERC4626CrossChainPort.sol#L322

```
function submitQueuedWithdrawBatch(address[] calldata users,
    address submissionRelayer, uint96 submissionRelayFee,
    address confirmationRelayer, uint96 confirmationRelayFee,
    Permit2 memory permit) external payable returns (uint16 withdrawBatchId)
    {
    // Expect caller to pay for total relay fee for withdraws.
    uint96 totalRelayFee = submissionRelayFee + confirmationRelayFee;
    PERMIT2.permitTransferFrom(ISignatureTransfer.PermitTransferFrom({
    permitted: ISignatureTransfer.TokenPermissions({ token: address(ASSET),
    amount: totalRelayFee }), nonce: permit.nonce, deadline:
    permit.deadline }), ISignatureTransfer.SignatureTransferDetails({ to:
    address(this), requestedAmount: totalRelayFee }), msg.sender,
    permit.signature);

322:    withdrawBatchId = _incrementWithdrawBatch();


    WithdrawBatch storage withdrawBatch = withdrawBatches[withdrawBatchId];
345:    withdrawBatch.totalSharesRedeemed = totalSharesToRedeem;
    withdrawBatch.totalAssetsDeposited = totalAssetsDeposited;
```

```
    _transferAssetsAcrossChain({ assets: totalRelayFee, relayer:
    submissionRelayer, relayFee: submissionRelayFee, deadline:
    uint32(block.timestamp + 1 hours), message:
    abi.encode(WithdrawSubmission({ submissionType: SubmissionType.Withdraw,
    withdrawBatchId: withdrawBatchId, totalSharesToRedeem:
    totalSharesToRedeem, confirmationRelayer: confirmationRelayer,
    confirmationRelayFee: confirmationRelayFee })) });
}
```

Now, suppose someone submits a `batch withdrawal` for `unlocked deposit rounds` using the `submitUnlockedDepositsWithdrawBatch` function:

- At `line 366`, the current `latestWithdrawBatchId` (3) is first assigned to `withdrawBatchId` and then increased by 1.
- As a result, `withdrawBatchId` remains 3, and the new shares are overwritten at `line 394`.
- ERC4626CrossChainPort.sol#L366

```
function submitUnlockedDepositsWithdrawBatch(uint16[]
    calldata depositRoundIds, address submissionRelayer,
    uint96 submissionRelayFee, address confirmationRelayer,
    uint96 confirmationRelayFee, Permit2 memory permit)
    external payable returns (uint16 withdrawBatchId) {
    // Expect caller to pay for total relay fee for withdraws.
    uint96 totalRelayFee = submissionRelayFee + confirmationRelayFee;
    PERMIT2.permitTransferFrom(ISignatureTransfer.PermitTransferFrom({
    permitted: ISignatureTransfer.TokenPermissions({ token: address(ASSET),
    amount: totalRelayFee }), nonce: permit.nonce, deadline:
    permit.deadline }), ISignatureTransfer.SignatureTransferDetails({ to:
    address(this), requestedAmount: totalRelayFee }), msg.sender,
    permit.signature);

366:    withdrawBatchId = latestWithdrawBatchId++;


    WithdrawBatch storage withdrawBatch = withdrawBatches[withdrawBatchId];
394:    withdrawBatch.totalSharesRedeemed = totalSharesToRedeem;
    withdrawBatch.totalAssetsDeposited = totalAssetsDeposited;

    // Transfer some assets to cover relaying submission and confirmation.
    _transferAssetsAcrossChain({ assets: totalRelayFee, relayer:
    submissionRelayer, relayFee: submissionRelayFee, deadline:
    uint32(block.timestamp + 1 hours), message:
    abi.encode(WithdrawSubmission({ submissionType: SubmissionType.Withdraw,
    withdrawBatchId: withdrawBatchId, totalSharesToRedeem:
```

```
    totalSharesToRedeem, confirmationRelayer: confirmationRelayer,
    confirmationRelayFee: confirmationRelayFee })) });
}
```

Obviously, these two `withdrawals` are distinct but share the same `withdrawBatchId`. As a result, one of them is accidentally deleted. When these two `withdrawals` are confirmed, the `rate` between the `shares` and `assets` differs. However, users in both `withdrawals` will use the latest `rate` because the previous result is overwritten.

- ERC4626CrossChainPort.sol#L406

```
function _onWithdrawConfirmed(uint16 withdrawBatchId,
    uint96 totalAssetsWithdrawn) internal {
    WithdrawBatch storage withdrawBatch = withdrawBatches[withdrawBatchId];

    // Take performance fee from assets withdrawn.
    uint96 fee;
    if (totalAssetsWithdrawn > withdrawBatch.totalAssetsDeposited) {
        uint96 gains = totalAssetsWithdrawn
    - withdrawBatch.totalAssetsDeposited;
        fee = gains.mulDiv(FEE_BPS, MAX_BPS).toUint96();
        ASSET.transfer(FEE_RECIPIENT, fee);
    }

    // Record remaining assets for users to claim.
    withdrawBatch.totalAssetsClaimable = totalAssetsWithdrawn - fee;
}
```

Recommendations:

```
function submitUnlockedDepositsWithdrawBatch(uint16[]
    calldata depositRoundIds, address submissionRelayer,
    uint96 submissionRelayFee, address confirmationRelayer,
    uint96 confirmationRelayFee, Permit2 memory permit)
    external payable returns (uint16 withdrawBatchId) {
-    withdrawBatchId = latestWithdrawBatchId++;
+    withdrawBatchId = _incrementWithdrawBatch();
}
```

PartyDAO: Use a consistent approach with prefix increment (e.g., _incrementWithdrawBatch) in both functions to ensure unique batch IDs. Patched in PR-18

Zenith: Verified

[H-7] Lack of token allowance when deposit to the vault in Adapter.sol

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: High

Target

- ERC4626CrossChainAdapter.sol

Description:

```
function _onDepositSubmitted(
      uint16 depositRoundId, uint96 totalAssetsToDeposit, address relayer,
    uint96 relayFee) internal {
      // Deposit assets into vault and get shares minted


      uint256 sharesMintedFromVault = VAULT.deposit({ assets:
    totalAssetsToDeposit, receiver: address(this) });

      // Send confirmation message back to port
      _sendMessageAcrossChain({ relayer: relayer, relayFee: relayFee,
    deadline: uint32(block.timestamp + 1 hours), message:
    abi.encode(ConfirmationType.Deposit, DepositConfirmation({
    confirmationType: ConfirmationType.Deposit, depositRoundId:
    depositRoundId, totalSharesMinted: uint96(sharesMintedFromVault) })) });
  }
```

The vault will be ERC4626 compatible.

Then the vault will requires approval before depositing the token to the vault. Otherwise, `VAULT.deposit` will revert with insufficient allowance.

The impact is that the deposit cannot be confirmed and the protocol's fund is locked.

Recommendations:

```
asset.approve(address(VAULT), totalAssetsToDeposit);
uint256 sharesMintedFromVault = VAULT.deposit({ assets:
    totalAssetsToDeposit, receiver: address(this) });
```

PartyDAO: Use SafeERC20.forceApprove() to give the ERC4626 vault approval to pull tokens from the adapter. Patched in PR-11

Zenith: Verified

[H-8] Refunded deposit is locked in the contract

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: High

Target

- ERC4626CrossChainPort.sol
- ERC4626CrossChainAdapter.sol

Description:

Current `fillDeadline` is 1 hour and according to the docs, the transaction will be reverted after this time.

> The deadline for the relayer to fill the deposit. After this destination chain timestamp, the fill will revert on the destination chain. Must be set between [currentTime, currentTime + fillDeadlineBuffer] where currentTime is block.timestamp on this chain or this transaction will revert.

tracking-events#expired-deposits

> In the meantime, we believe its accurate to state that expired deposits will be refunded approximately 90 minutes after their fillDeadline. Expired deposits are sent to the depositor address on the originChainId

According to the doc, the expired deposit will be refunded to the depositor address.

However, there is no function to handle the refunded token, and these token are locked.

This issue exists in both `Port` contract and `Adapter` contract.

Recommendations:

Consider add a function to sweep the fund from the contract.

Ensure that the only a timelock smart contract can trigger such function.

PartyDAO: In PR-10, the function sweepToken is added in both Port and adapter contract

to ensure owner and sweep the reverted refunded token.

Zenith: Verified

4.2 Medium Risk

A total of 4 medium risk findings were identified.

[M-1] Users can lose their shares without claiming the actual assets in the claimUnlockedDepositWithdraw function

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Low

Target

- ERC4626CrossChainPort.sol

Description:

Users can `claim` their `assets` from `deposit rounds`. It is important to note that these `rounds` should be included in a `withdrawal batch`, and this check is performed at `line 448`.

- ERC4626CrossChainPort.sol#L455

```
function claimUnlockedDepositWithdraw(uint16[] calldata depositRoundIds,
    uint96[] calldata shares, address recipient)
    external returns (uint96 totalAssetsClaimed) {
    require(depositRoundIds.length == shares.length, "Array lengths must
    match");
    for (uint16 i; i < depositRoundIds.length; i++) {
        uint16 depositRoundId = depositRoundIds[i];
        DepositRound storage depositRound = depositRounds[depositRoundId];
448:        require(depositRound.submittedInWithdrawBatchId > 0, "Unlocked
    deposit not withdrawn");
        uint96 assetsDeposited = depositRound.assetsDepositedBy[msg.sender];
        uint96 sharesMinted
    = depositRound.totalSharesMinted.mulDiv(assetsDeposited,
    depositRound.totalAssetsDeposited).toUint96();
        uint96 sharesRedeemable = sharesMinted
    - depositRound.sharesRedeemedBy[msg.sender];
        uint96 sharesToRedeem = sharesRedeemable < shares[i] ?
    sharesRedeemable : shares[i];
        if (sharesToRedeem > 0) {
```

```
            WithdrawBatch storage withdrawBatch
    = withdrawBatches[depositRound.submittedInWithdrawBatchId];
455:            uint96 assetsToClaim
    = withdrawBatch.totalAssetsClaimable.mulDiv(sharesToRedeem,
    withdrawBatch.totalSharesRedeemed).toUint96();
            totalAssetsClaimed += assetsToClaim;
            depositRound.sharesRedeemedBy[msg.sender] += sharesToRedeem;
        }
    }
    require(totalAssetsClaimed > 0, "No assets to claim");
    ASSET.transfer(recipient, totalAssetsClaimed);
}
```

This ensures that the `request` to `withdraw assets` related to the `shares` of these `rounds` is sent to the `Adapter`. However, this does not guarantee that these `withdrawals` have been executed.

There is always a gap between the `request` and the `actual withdrawal`. When the `withdrawal` is executed, the `totalAssetsClaimable` of the `withdrawal batch` is updated to a non-zero value, allowing users to `claim` the real `assets`.

However, at `line 455`, users can claim their `assets` without checking whether `totalAssetsClaimable` is non-zero. Regardless of this, the `sharesRedeemedBy` value for the user increases.

Recommendations:

```
function claimUnlockedDepositWithdraw(uint16[] calldata depositRoundIds,
    uint96[] calldata shares, address recipient)
    external returns (uint96 totalAssetsClaimed) {
    require(depositRoundIds.length == shares.length, "Array lengths must
    match");
    for (uint16 i; i < depositRoundIds.length; i++) {
        if (sharesToRedeem > 0) {
            WithdrawBatch storage withdrawBatch
    = withdrawBatches[depositRound.submittedInWithdrawBatchId];
.           require
. (withdrawBatch.totalAssetsClaimable != 0, "Invalid claim");
            uint96 assetsToClaim
    = withdrawBatch.totalAssetsClaimable.mulDiv(sharesToRedeem,
    withdrawBatch.totalSharesRedeemed).toUint96();
            totalAssetsClaimed += assetsToClaim;
            depositRound.sharesRedeemedBy[msg.sender] += sharesToRedeem;
        }
    }
```

```
}
```

PartyDAO: Resolved with PR-20

Zenith: Verified

[M-2] Revert condition in external call Vault#deposit and Vault#redeem is not handled

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- ERC4626CrossChainAdapter.sol

Description:

cross-chain-actions-integration-guide

> If the recipient contract's handleV3AcrossMessage function reverts when message , tokenSent, amount, and relayer are passed to it, then the fill on destination will fail and cannot occur. In this case the deposit will expire when the destination SpokePool timestamp exceeds the deposit fillDeadline timestamp, the depositor will be refunded on the originChainId. Ensure that the depositor address on the origin SpokePool is capable of receiving refunds.

Assume that a standard ERC4626 is used for vault contract,

the `VAULT.deposit` and `VAULT.redeem` external call can revert.

```
function deposit(uint256 assets, address receiver)
    public virtual returns (uint256) {
    uint256 maxAssets = maxDeposit(receiver);
    if (assets > maxAssets) {
        revert ERC4626ExceededMaxDeposit(receiver, assets, maxAssets);
    }
```

Then the refund will be sent to depositor address and the depositor address lacks function to handle refund.

Recommendations:

Try catch the vault related external call and add a function to sweep the fund from adapter. Only a timelock contract should be able to sweep the refund to ensure security.

PartyDAO: Resolved with PR-10

Zenith: Verified

[M-3] Minimum relayer fee should not be proportional to total deposit amount in the Port.sol

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- ERC4626CrossChainPort.sol

Description:

if the cross-chain transaction is not submitted by PRIORITIZED_SUBMITTER, the code enforces a minimum amount of total relayer fee.

```
if (msg.sender ≠ PRIORITIZED_SUBMITTER) {
    // Prevent submitting withdraw batch with too low of a relay fee to avoid
    DoS attacks.
    uint96 minRelayFee = totalAssetsDeposited.mulDiv(MIN_RELAY_FEE_BPS,
    MAX_BPS).toUint96();
    require(totalRelayFee >= minRelayFee, "Relay fee too low");
}
```

However, the Minimum relayer fee should not be proportional to total deposit amount.

Assume that the MIN_RELAY_FEE_BPS is 1%.

If the total deposit for a round is 1 million (total deposit is large), the total relayer fee becomes 10K, which is clearly too high for the caller that pay the relayer fee.

if the total deposit for a round is 1 dollar, the total relayer fee becomes 1 / 100 dollar, which is clearly to low and the relayer lacks incentive to relay the transaction.

Recommendations:

Enforce a minimum submission relayer fee and confirmation relayer fee instead of let minimum relayer fee should not be proportional to total deposit amount.

PartyDAO: Resolved with PR-9

Zenith: Verified.

[M-4] User can set confirmation relayer fee or submission relayer fee to 0 to disincentivize relayer from filling the cross-chain message.

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- ERC4626CrossChainPort.sol

Description:

After the priorityWindow passed, anyone can execute the cross-chain deposit / withdraw request.

```
function submitDepositRound(
    address submissionRelayer,
    uint96 submissionRelayFee,
    address confirmationRelayer,
    uint96 confirmationRelayFee,
    Permit2 memory permit)
 external returns (uint16 depositRoundId) {

    depositRoundId = latestDepositRoundId;
    DepositRound storage depositRound = depositRounds[depositRoundId];

    require(depositRound.submittedAt == 0, "Deposit round already
    submitted");
    require(depositRound.totalAssetsDeposited > 0, "No assets to submit");

    uint96 totalRelayFee = submissionRelayFee + confirmationRelayFee;

    if (msg.sender != PRIORITIZED_SUBMITTER) {
        uint96 minRelayFee
    = depositRound.totalAssetsDeposited.mulDiv(MIN_RELAY_FEE_BPS,
    MAX_BPS).toUint96();
        require(totalRelayFee >= minRelayFee, "Relay fee too low");

        uint32 priorityWindowEndsAt = lastDepositRoundSubmittedAt
    + SUBMIT_DEPOSIT_ROUND_PRIORITY_WINDOW;
```

```
        require(block.timestamp > priorityWindowEndsAt, "Priority submitter
    window active");
    }
```

If the msg.sender is not PRIORITIZED_SUBMITTER, the code validate the totalRelayFee exceed minRelayFee

However, the code does not validate the min amount of submissionRelayFee and confirmationRelayFee.

Malicious executor can set either submissionRelayFee or confirmationRelayFee to 0.

Example:

100% of the total fee is submissionRelayFee.The confirmationRelayFee is 0.

Then the confirmationRelayer has no incentive to execute the confirmation cross-chain message OR the confirmationRelayer has to pay the gas fee to execute the cross-chain message with no relayer fee.

Recommendations:

Validate the min amount of submission fee and relayer fee is offered.

PartyDAO: Patched indirectly by removing need for relay fee validation for untrusted callers in PR-9

Zenith: Verified

4.3 Low Risk

A total of 4 low risk findings were identified.

[L-1] There is no function to update approved relayers in the Port

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- ERC4626CrossChainPort.sol

Description:

`Relayers` play an important role in the protocol, and there will be a need to update `approved relayers`.

The `Adapter` has such a function, but it is missing in the `Port`.

- ERC4626CrossChainAdapter.sol#L79

```
function setApprovalForRelayers(address[] calldata relayers, bool[] calldata
    isApproved) external onlyOwner {
    require(relayers.length == isApproved.length, "Array lengths must
    match");
    for (uint256 i; i < relayers.length; i++) isApprovedRelayer[relayers[i]]
    = isApproved[i];
}
```

Recommendations:

Add the function to the `Port` to allow the admin to update the approved relayer address.

PartyDAO: Added the function setApprovalForRelayers in ERC4626CrossChainPort.sol

Zenith: Verified

[L-2] User can frontrun the submission by canceling withdraw to force `submitQueuedWithdrawBatch` to revert

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Medium

Target

- ERC4626CrossChainPort.sol

Description:

When user submit batch withdraw, the code will revert if the user cancel the withdraw.

```
for (uint256 i; i < users.length; i++) {
    address user = users[i];
    Withdraw storage pendingWithdraw = pendingWithdraws[user];
    require(pendingWithdraw.submittedInWithdrawBatchId == 0, "Withdraw
    already submitted");
    require(pendingWithdraw.validAt != 0 && pendingWithdraw.validAt <
    block.timestamp, "Invalid withdraw"); // @audit continue
    totalSharesToRedeem += pendingWithdraw.sharesRedeemed;
    totalAssetsDeposited += pendingWithdraw.assetsDeposited;
    pendingWithdraw.submittedInWithdrawBatchId = withdrawBatchId;
}
```

Note the line of code:

```
require(pendingWithdraw.validAt != 0 && pendingWithdraw.validAt <
    block.timestamp, "Invalid withdraw"); // @audit continue
```

Then a user can frontrun the submission by canceling withdraw to force `submitQueuedWithdrawBatch` to revert with error `Invalid withdraw`

```
function cancelWithdraw() external {
    Withdraw storage pendingWithdraw = pendingWithdraws[msg.sender];
    require(pendingWithdraw.submittedInWithdrawBatchId == 0, "Withdraw
    already submitted");
    delete pendingWithdraw.validAt;
```

```
}
```

Recommendations:

```
for (uint256 i; i < users.length; i++) {
            address user = users[i];
            Withdraw storage pendingWithdraw = pendingWithdraws[user];
            require(pendingWithdraw.submittedInWithdrawBatchId == 0,
    "Withdraw already submitted");

    require(pendingWithdraw.validAt ≠ 0 && pendingWithdraw.validAt <
                block.timestamp, "Invalid withdraw"); // @audit continue
          if (pendingWithdraw.validAt == 0 || pendingWithdraw.validAt ≥
                block.timestamp) { continue; }
```

PartyDAO: Acknowledged.

[L-3] Consider add setter for immutable variable

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- ERC4626CrossChainPort.sol

Description:

```
uint256 public immutable ADAPTER_CHAIN_ID; // @audit avoid hardcode
 ERC20 public immutable ADAPTER_CHAIN_ASSET; // @audit
 ISpokePool public immutable ACROSS;
```

The protocol hardcode the ADAPTER_CHAIN_ID and ADAPTER_CHAIN_ASSET and ACROSS.

If these variable changes from the Across protocol side, the cross-chain deposit request will fail.

Recommendations:

Add setter function to update `ADAPTER_CHAIN_ID` and `ADAPTER_CHAIN_ASSET` and `ACROSS`

PartyDAO: Resolved with PR-13

Zenith: Verified.

[L-4] Consider use uint256 instead of uint16 for round tracking

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- ERC4626CrossChainPort.sol

Description:

The code use uint16 to track deposit round.

```
mapping(uint16 ⇒ DepositRound) public depositRounds;
```

The max number of uint16 is 65535.

After 65535 rounds, no deposit and no withdraw round can be made.

Recommendations:

```
mapping(uint16 ⇒ DepositRound) public depositRounds;
    /// @dev ... May either be a group of withdraws queued for individual
    depositors (submitQueuedWithdrawBatch) or a
    /// single withdraw of an entire deposit round for multiple depositors at
    once
    /// (submitUnlockedDepositsWithdrawBatch).
mapping(uint16 ⇒ WithdrawBatch) public withdrawBatches;
mapping(uint256⇒ DepositRound) public depositRounds;
mapping(uint256 ⇒ WithdrawBatch) public withdrawBatches;
```

PartyDAO: Resolved with PR-14

Zenith: Verified.

4.4 Informational

A total of 4 informational findings were identified.

[I-1] Users may lose some assets due to the fee

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- ERC4626CrossChainPort.sol

Description:

Suppose the current `share price` in the `vault` is 1. `User A` deposits `100 assets` and receives `100 shares`. After some time, the `share price` increases to 2. `User B` then deposits `200 assets` and receives `100 shares`. These `assets` enter the `withdrawal batch`.

Finally, `400 assets`, equivalent to `200 shares`, will be withdrawn. If the `fee` is `10%`, `360 assets` will be assigned to the users. Since both users have the same number of `shares`, they will each receive `180 assets`.

As a result, `User A` gains more, but `User B` loses `20 assets`. This may be an unavoidable and intended situation.

PartyDAO: Resolved with PR-8

Zenith: Verified

[I-2] The check for sharesRedeemed can be added to the validateWithdraw function

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- ERC4626CrossChainPort.sol

Description:

There is no check to determine whether `sharesRedeemed` is 0 in the `validateWithdraw` function.

- ERC4626CrossChainPort.sol#L294-L299

```
function validateWithdraw(uint32 validAt) external {
    require(validAt > 0, "Invalid validation timestamp");
    Withdraw storage pendingWithdraw = pendingWithdraws[msg.sender];
    require(pendingWithdraw.submittedInWithdrawBatchId == 0, "Withdraw
    already submitted");
    pendingWithdraw.validAt = validAt;
}
```

If the `validAt` value is updated to a non-zero value, the `pending withdrawal` for the user can be included in the `withdrawal batch`, even if it is actually empty.

- ERC4626CrossChainPort.sol#L333

```
function submitQueuedWithdrawBatch(
    address[] calldata users,
    address submissionRelayer,
    uint96 submissionRelayFee,
    address confirmationRelayer,
    uint96 confirmationRelayFee,
    Permit2 memory permit
) external payable returns (uint16 withdrawBatchId) {
    uint96 totalSharesToRedeem;
    uint96 totalAssetsDeposited;
    for (uint256 i; i < users.length; i++) {
```

```
        address user = users[i];
        Withdraw storage pendingWithdraw = pendingWithdraws[user];
        require(pendingWithdraw.submittedInWithdrawBatchId == 0, "Withdraw
    already submitted");
@->         require(pendingWithdraw.validAt != 0 && pendingWithdraw.validAt <
    block.timestamp, "Invalid withdraw");
        totalSharesToRedeem += pendingWithdraw.sharesRedeemed;
        totalAssetsDeposited += pendingWithdraw.assetsDeposited;
@->         pendingWithdraw.submittedInWithdrawBatchId = withdrawBatchId;
    }
}
```

Consequently, the `submittedInWithdrawBatchId` is updated, which means that queueing a withdrawal for this user will be blocked until the `withdrawal batch` that includes the user's empty `withdrawal request` is confirmed.

Recommendations:

```
function validateWithdraw(uint32 validAt) external {
    require(validAt > 0, "Invalid validation timestamp");
    Withdraw storage pendingWithdraw = pendingWithdraws[msg.sender];
    require(pendingWithdraw.submittedInWithdrawBatchId == 0, "Withdraw
    already submitted");
  require(pendingWithdraw.sharesRedeemed != 0, "");
    pendingWithdraw.validAt = validAt;
}
```

PartyDAO: Resolved with PR-17

Zenith: Verified

[I-3] The 1-hour deadline should be adjusted

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- ERC4626CrossChainPort.sol

Description:

Currently, the `deadline` is fixed at `1 hour`.

This `deadline` should be less than the `fillDeadlineBuffer` in `ACROSS`.

The deployed `fillDeadlineBuffer` in `WorldChain` is now `6 hours`, so it is acceptable.

However, in cases where the `1-hour deadline` is not sufficient for successful transfers between `chains`, it is better to dynamically adjust it.

Recommendations:

Add a function to set the deadline instead of hardcode the deadline to 1 hour.

PartyDAO: Resolved with PR-13. Added the setter delayUntilDeadline

Zenith: Verified

[I-4] The addresses for the contracts on the destination chain can be stored

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- ERC4626CrossChainPort.sol
- ERC4626CrossChainAdapter.sol

Description:

The `Port` exists on the `source chain`, and the `Adapter` exists on the `destination chain`. The `Adapter` should be stored in the `Port` contract. However, the `Adapter` contract may not exist on the `source chain`.

- ERC4626CrossChainPort.sol#L90

```
contract ERC4626CrossChainPort is Ownable, Multicall {
    ERC4626CrossChainAdapter public immutable ADAPTER;
    ERC20 public immutable ADAPTER_CHAIN_ASSET;
}
```

Therefore, the address of the `Adapter` on the `destination chain` can be stored in the `Port contract`. The same issue exists for the `target asset` and the `Port` in the `Adapter` contract.

Recommendations:

Store `addresses` instead of `contracts`.

PartyDAO: Acknowledged