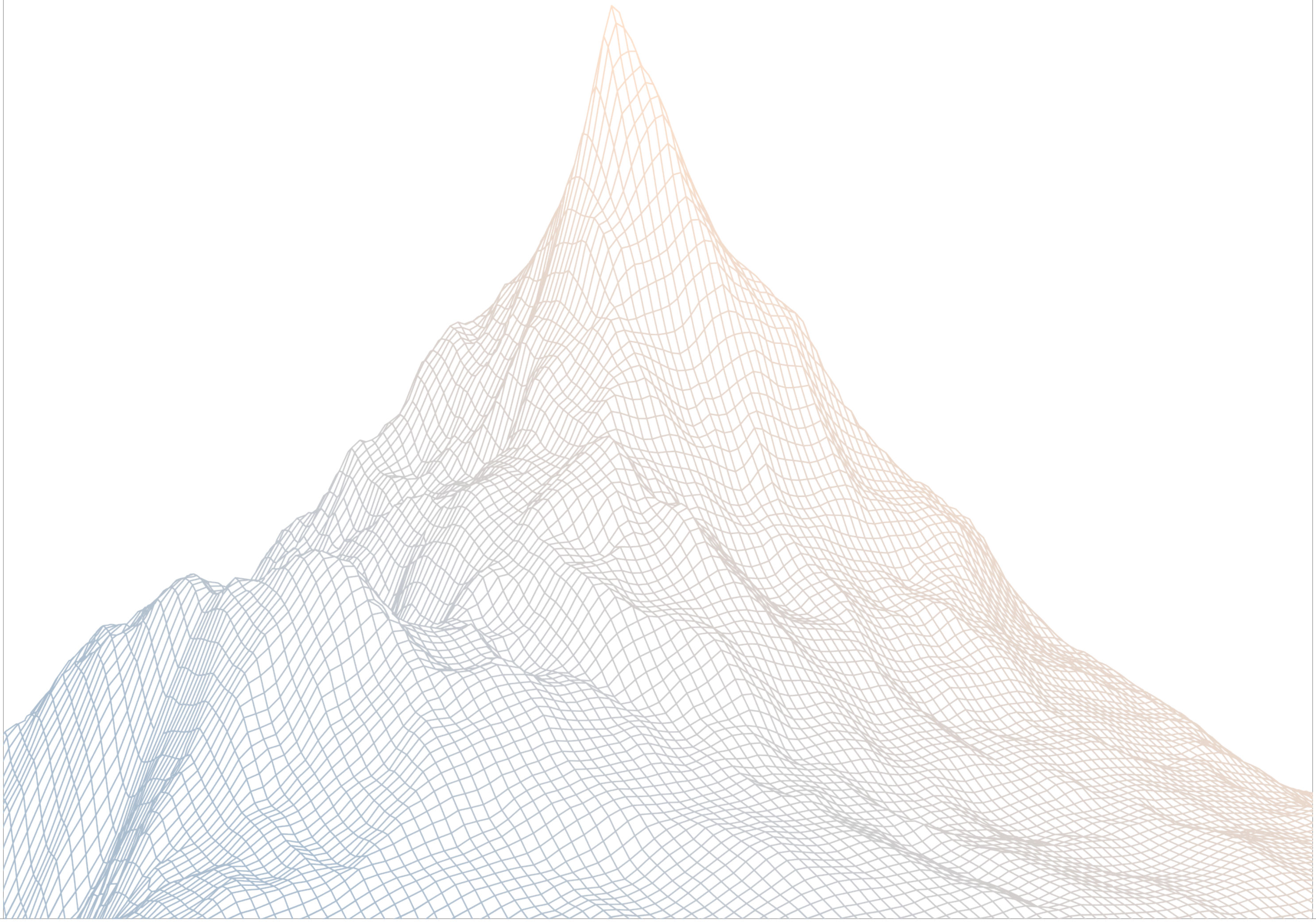
Zenith

Doppler

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES:	May 30th to May 31th, 2025
AUDITED BY:	ethersky ladboy

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

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 Doppler

XRP and the XRP Ledger (XRPL) ecosystem are entering another phase of growth. Previously, the on-chain ecosystem was primarily shaped around institutions. However, recent changes such as the emergence of various sidechains, the introduction of AMM DEX pools, and the issuance of RLUSD are driving significant changes to the ecosystem. Such changes are driving the on-chain activities on the XRPL ecosystem.

At the heart of these changes, Doppler Finance is introducing a new paradigm, “XRPfi,” enabling holders to earn yields with their XRP or assets within the XRPL ecosystem. The yield opportunities proposed by XRPfi aim to unlock new growth potential for XRP, leveraging yield as a driving force.

2.2 Scope

The engagement involved a review of the following targets:

Target	vault-public
Repository	https://github.com/girin-app/vault-public
Commit Hash	`50de72d86e769799aa1c72ba0e24dd7c66e5d2a4`
Files	`contracts/*`

2.3 Audit Timeline

May 30, 2025	Audit start
May 30, 2025	Audit end
June 4, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	1
Low Risk	3
Informational	4
Total Issues	**9**

3 Findings Summary

ID	Description	Status
H-1	In some cases, users are unable to withdraw their funds	Resolved
M-1	Users cannot claim their funds if the undelegated funds are redelegated using the delegate function	Resolved
L-1	A check to ensure sufficient interests amount is available should be added to the updateYield function	Acknowledged
L-2	A timestamp check should be included in the undelegate function	Resolved
L-3	The calculation of the removed share should be done in favor of the Vault	Resolved
I-1	The check should be updated in the _undelegate function	Resolved
I-2	A check should be added to ensure that the new treasury holds enough tokens to cover both the total supply and total interest in the setTreasury function	Resolved
I-3	Consider validating the max yield range in Vault.sol	Resolved
I-4	Gateway contract does not need to have an owner	Resolved

4 Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] In some cases, users are unable to withdraw their funds

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

Target

- Vault.sol

Description:

Suppose `User 1` deposits `500` using the `deposit()` function.

- Vault.sol#L207-L212

```
function deposit(uint256 amount) external whenNotPaused {
    require(amount > 0, "Amount must be greater than 0");
    require(totalSupply + amount <= depositCap, "Deposit cap exceeded");

    uint256 xAmount = (totalXSupply == 0) ? amount : amount * totalXSupply
    / (totalSupply + totalInterest);

    depositBalance[msg.sender] += amount;
    depositXBalance[msg.sender] += xAmount;
    totalSupply += amount;
    totalXSupply += xAmount;

    IERC20(token).safeTransferFrom(msg.sender, treasury, amount);
    totalDelegatedBalance += amount;

    emit Deposited(msg.sender, amount, xAmount);
}
```

The initial values are as follows.

```
xAmount = 500
depositBalance[User 1] = 500
depositXBalance[User 1] = 500
totalSupply = 500
totalXSupply = 500
totalInterest = 0
```

Later, `100 in interest` is generated.

- Vault.sol#L360

```
function updateYield(uint256 amount) external onlyOperator whenNotPaused {
    require(amount > 0, "Amount must be greater than 0");

    totalInterest += amount;
}
```

The updated values are shown below.

```
totalInterest = 100
```

Since the `100 interest` belongs to `User 1`, he should be able to `withdraw` a total of `600`.

Then, `User 2` deposits `300` using the `deposit()` function. The current values for both users are updated accordingly.

```
xAmount = 300 * 500 / 600 = 250
depositBalance[User 1] = 500
depositXBalance[User 1] = 500
depositBalance[User 2] = 300
depositXBalance[User 2] = 250
totalSupply = 800
totalXSupply = 750
totalInterest = 100
```

Now, `User 1` attempts to `withdraw` `600` using the `withdraw()` function.

- Vault.sol#L266-L273

```
function withdraw(uint256 amount) external whenNotPaused {
    uint256 userXBalance = depositXBalance[msg.sender];
263:    uint256 maxWithdrawAmount = userXBalance * (totalSupply
    + totalInterest) / totalXSupply;
264:    require(amount <= maxWithdrawAmount, "Insufficient balance");
```

```
    uint256 xAmount = amount * totalXSupply / (totalSupply + totalInterest);
267:    uint256 principalAmount = amount * totalSupply / (totalSupply
    + totalInterest);
    uint256 interestAmount = amount - principalAmount;

    depositXBalance[msg.sender] -= xAmount;
    totalXSupply -= xAmount;

273:    depositBalance[msg.sender] -= principalAmount;
    totalSupply -= principalAmount;
    totalInterest -= interestAmount;
}
```

- At **line 263**, `maxWithdrawAmount` is correctly calculated as `500 * 900 / 750 = 600`, so the check at **line 264** passes.
- However, at **line 267**, `principalAmount` is computed as `600 * 800 / 900 = 533`, which is **greater than** `depositBalance[User1] = 500`.
- As a result, the transaction is **reverted** at **line 273**.

The root cause of this issue is the **incorrect update of *`depositBalance`***.

Recommendations:

Tracking `depositBalance` accurately is difficult, and there appears to be no practical use case for this value. Therefore, it could be safely ignored.

Doppler Finance: Fixed in the commit @4d24699...

Zenith: Verified.

4.2 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] Users cannot claim their funds if the undelegated funds are redelegated using the delegate function

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

Target

- Vault.sol

Description:

When users `deposit` funds, the funds are transferred directly to the `treasury`.

- Vault.sol#L214

```
function deposit(uint256 amount) external whenNotPaused {
    IERC20(token).safeTransferFrom(msg.sender, treasury, amount);
    totalDelegatedBalance += amount;

    emit Deposited(msg.sender, amount, xAmount);
}
```

If a user wants to `withdraw` their funds, those funds must first be `undelegated`. This means the funds need to be transferred back from the `treasury` to the `vault` before they can be claimed.

- Vault.sol#L250

```
function _undelegate(uint256 targetTimestamp) private {
    // Transfer tokens from treasury to this contract
    IERC20(token).safeTransferFrom(treasury, address(this), totalAmount);
    totalUndelegatedBalance += totalAmount;
}
```

Currently, there is no mechanism to delete `withdrawal` requests. However, the `owner` can `delegate` the funds to the `treasury` using the `delegate` function.

- Vault.sol#L225

```
function delegate(uint256 amount) external onlyOwner whenNotPaused {
    IERC20(token).safeTransfer(treasury, amount);
    totalDelegatedBalance += amount;

    emit Delegated(amount);
}
```

Once this occurs, users will be unable to claim their tokens due to insufficient funds in the `vault`.

Maybe the `delegate` function might be used to delegate untracked funds (i.e., funds that were not undelegated for withdrawals). If that's the intended use case, then these funds should be added to `totalInterest` to ensure they are properly tracked. In either case, the impact of this behavior is high.

Recommendations:

We can track the undelegated amounts from the `treasury` that are reserved for `withdrawals`, and in the `delegate` function, consider only the difference between the current `vault balance` and these tracked amounts.

Doppler Finance: Fixed in the commit @eccc347...

Zenith: Verified. Removed the `delegate` function.

4.3 Low Risk

A total of 3 low risk findings were identified.

[L-1] A check to ensure sufficient interests amount is available should be added to the updateYield function

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

Target

- Vault.sol

Description:

`Interest` is generated through the `updateYield` function, and both the `total supply` and `total interest` are reserved in the `treasury`. However, the `updateYield` function does not include a check to ensure that the `treasury` actually holds these increased interest amount.

- Vault.sol#L360

```
function updateYield(uint256 amount) external onlyOperator whenNotPaused {
    // Update total interest
    totalInterest += amount;
}
```

Recommendations:

```
function updateYield(uint256 amount) external onlyOperator whenNotPaused {
    // Update total interest
    totalInterest += amount;
+^^Irequire(IERC20(token).balanceOf(treasury) >= totalSupply
    + totalInterest, "");
}
```

Doppler Finance: Acknowledged.

We do not keep assets deposited in the treasury at all times. Assets in the treasury will be transferred elsewhere after being deposited, and since there is a process in place to replenish the treasury with the amount necessary for withdrawals on a 7-day basis, it was intentional not to check the balance at the time of updating the yield. We were aware of this, but it seems that more attention is needed when proceeding with the process.

[L-2] A timestamp check should be included in the undelegate function

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

Target

- Vault.sol

Description:

Funds can only be `undelegated` at least `7 days` after a `withdrawal request` is made. This rule is enforced in the `undelegate` function, which can be called by the `operator`.

- Vault.sol#L231

```
function undelegate() external onlyOperator whenNotPaused {
    uint256 targetTimestamp = getTargetUndelegateTimestamp(block.timestamp);
    _undelegate(targetTimestamp);
}
```

However, the `owner` can bypass this restriction by calling the `undelegate` function with an arbitrary `timestamp`, allowing `undelegation` to occur earlier than the required `7-day` period.

- Vault.sol#L236

```
function undelegate(uint256 timestamp) external onlyOwner whenNotPaused {
    uint256 targetTimestamp = getTargetUndelegateTimestamp(timestamp);
    _undelegate(targetTimestamp);
}
```

Recommendations:

```
function undelegate(uint256 timestamp) external onlyOwner whenNotPaused {
+^^Irequire(timestamp <= block.timestamp, "");
    uint256 targetTimestamp = getTargetUndelegateTimestamp(timestamp);
    _undelegate(targetTimestamp);
}
```

Doppler Finance: Fixed in the commit @eccc347...

Zenith: Verified.

[L-3] The calculation of the removed share should be done in favor of the Vault

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

Target

- Vault.sol

Description:

When users `withdraw` funds, the removed `shares` are calculated as follows.

- Vault.sol#L266

```
function withdraw(uint256 amount) external whenNotPaused {
    require(amount > 0, "Amount must be greater than 0");

    uint256 userXBalance = depositXBalance[msg.sender];
    uint256 maxWithdrawAmount = userXBalance * (totalSupply + totalInterest)
    / totalXSupply;
    require(amount <= maxWithdrawAmount, "Insufficient balance");

    uint256 xAmount = amount * totalXSupply / (totalSupply + totalInterest);

    depositXBalance[msg.sender] -= xAmount;
}
```

However, this calculation favors the user rather than the `Vault`. In most protocols, all calculations are designed to favor the `Vault`. This can lead to unexpected outcomes, such as `withdrawal reverts` due to small discrepancies (e.g., dust amounts).

Recommendations:

```
function withdraw(uint256 amount) external whenNotPaused {
    require(amount > 0, "Amount must be greater than 0");

    uint256 userXBalance = depositXBalance[msg.sender];
    uint256 maxWithdrawAmount = userXBalance * (totalSupply + totalInterest)
    / totalXSupply;
    require(amount <= maxWithdrawAmount, "Insufficient balance");

-     uint256 xAmount = amount * totalXSupply / (totalSupply + totalInterest);
+^^Iuint256 xAmount = (amount * totalXSupply + totalSupply + totalInterest
    - 1) / (totalSupply + totalInterest);

    depositXBalance[msg.sender] -= xAmount;
}
```

Doppler Finance: Fixed in the commit @eccc347...

Zenith: Verified.

4.4 Informational

A total of 4 informational findings were identified.

[I-1] The check should be updated in the _undelegate function

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

Target

- Vault.sol

Description:

In the `_undelegate` function, if the `withdrawPerDay` value is 0 for a specific timestamp, it is assumed that there are no `withdrawal` amounts to `release` for that day.

- Vault.sol#L243

```
function _undelegate(uint256 targetTimestamp) private {
    require(!withdrawReleased[targetTimestamp], "Withdrawals already
    released for this unit");
    require(withdrawPerDay[targetTimestamp] > 0, "No withdrawals to release
    for this unit");

    uint256 principalAmount = withdrawPerDay[targetTimestamp];
    uint256 interestAmount = withdrawInterestPerDay[targetTimestamp];
}
```

However, in some rare cases, `withdrawInterestPerDay` is not 0 while `withdrawPerDay` is 0, which leads to inconsistent behavior.

Recommendations:

```
function _undelegate(uint256 targetTimestamp) private {
    require(!withdrawReleased[targetTimestamp], "Withdrawals already
    released for this unit");
-    require(withdrawPerDay[targetTimestamp] > 0, "No withdrawals to release
    for this unit");
+^^Irequire(withdrawPerDay[targetTimestamp]
    + withdrawInterestPerDay[targetTimestamp] > 0, "No withdrawals to
    release for this unit");

    uint256 principalAmount = withdrawPerDay[targetTimestamp];
    uint256 interestAmount = withdrawInterestPerDay[targetTimestamp];
}
```

Doppler Finance: Fixed in the commit @eccc347...

Zenith: Verified.

[I-2] A check should be added to ensure that the new treasury holds enough tokens to cover both the total supply and total interest in the setTreasury function

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

Target

- Vault.sol

Description:

A new `treasury` can be set using the `setTreasury` function.

- Vault.sol#L180

```
function setTreasury(address _treasury) external onlyOwner {
    require(_treasury ≠ address(0), "Invalid address");
    treasury = _treasury;
    emit TreasuryUpdated(_treasury);
}
```

However, there is no check to verify whether the new `treasury` contains enough tokens.

Recommendations:

```
function setTreasury(address _treasury) external onlyOwner {
    require(_treasury ≠ address(0), "Invalid address");
    treasury = _treasury;
+^^Irequire(IERC20(token).balanceOf(_treasury) >= totalSupply
    + totalInterest, "");
    emit TreasuryUpdated(_treasury);
}
```

Doppler Finance: Fixed in the commit @eccc347...

Zenith: Verified.

[I-3] Consider validating the max yield range in `Vault.sol`

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

Target

- Vault.sol

Description:

```
uint256 public yieldMaxRate; // Max yield rate in basis points (e.g., 300 =
    3%)
```

By default, the `yieldMaxRate` is 3%, as the comment said, 300 is 3%, then the owner should not set the max yield rate to more than 100%.

```
function setYieldMaxRate(uint256 _rate) external onlyOwner {
     require(_rate > 0, "Rate must be greater than 0");
     yieldMaxRate = _rate;
     emit YieldMaxRateUpdated(_rate);
 }
```

Recommendations:

```
   function setYieldMaxRate(uint256 _rate) external onlyOwner {
        require(_rate > 0, "Rate must be greater than 0");
+     require(_rate <= 10000, "Rate must be greater than 0");
        yieldMaxRate = _rate;
        emit YieldMaxRateUpdated(_rate);
    }
```

Doppler Finance: Fixed in the commit @eccc347...

Zenith: Verified.

[I-4] `Gateway` contract does not need to have an owner

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

Target

- Gateway.sol

Description:

```
contract Gateway is Ownable {

    constructor(address _owner) Ownable(_owner) {}
    function claimAll(address user, address[] calldata vaults) external {
        require(user ≠ address(0), "Invalid user");
        for (uint256 i = 0; i < vaults.length; i++) {
            address vault = vaults[i];
            IVault(vault).claimBehalf(user);
        }
    }
}
```

The Gateway can be permissionlessly used to batch call `claimBehalf` and does not need an owner.

Recommendations:

```diff
- contract Gateway is Ownable {
+ contract Gateway {
```

Doppler Finance: Fixed in the commit @eccc347...

Zenith: Verified.