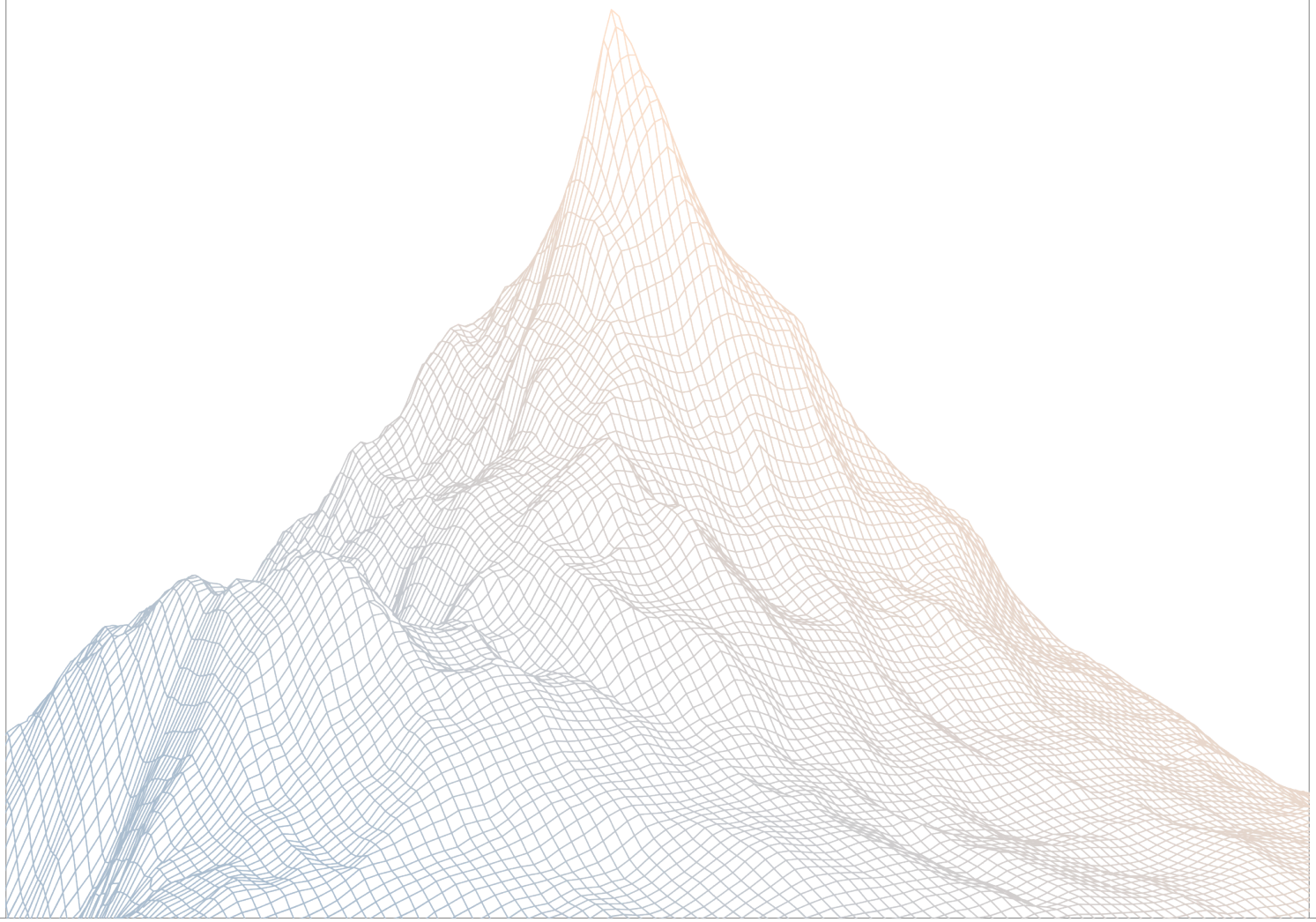
Zenith

Forte

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: October 24th to October 25th, 2025

AUDITED BY: SpicyMeatball
cccz

Contents

1 Introduction **2**

1.1 About Zenith 3

1.2 Disclaimer 3

1.3 Risk Classification 3

2 Executive Summary **3**

2.1 About Forte 4

2.2 Scope 4

2.3 Audit Timeline 5

2.4 Issues Found 5

3 Findings Summary **5**

4 Findings **6**

4.1 Medium Risk 7

4.2 Low Risk 9

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 Forte

Our mission is to enable the development of thriving economies built on blockchain. We are a group of deeply technical people who specialize in computer science, math, engineering, economics, and finance. We believe that blockchain-based technologies will cause important disruptions across all sectors of the current economy and we are building tools to accelerate this evolution.

2.2 Scope

The engagement involved a review of the following targets:

Target	liquidity-base
Repository	https://github.com/Forte-Service-Company-Ltd/liquidity-base
Commit Hash	`e65ee2644f50649a95b74b02032025d3bf5ef54e`
Files	`Review changes in the Diff`

Target	liquidity-altbc
Repository	https://github.com/Forte-Service-Company-Ltd/liquidity-altbc
Commit Hash	`4c365ab40df0bbb740df950531c857693308dc91`
Files	`Review changes in the Diff`

2.3 Audit Timeline

October 24, 2025	Audit start
October 25, 2025	Audit end
October 28, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	1
Informational	0
Total Issues	**2**

3

Findings Summary

ID	Description	Status
M-1	Unsafe transferFrom() may make the deployer not have to provide the initial liquidity	Resolved
L-1	LPToken needs to disable renounceOwnership()	Resolved

4

Findings

4.1 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] Unsafe `transferFrom()` may make the deployer not have to provide the initial liquidity

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

Target

- ALTBCFactory.sol#L72-L73

Description:

When the deployer creates a pool, the deployer needs to provide the initial xToken as the initial liquidity.

```
function createPool(
    address _xToken,
    address _yToken,
    uint16 _lpFee,
    ALTBCInput memory _tbcInput,
    uint256 _xAdd,
    uint256 _wInactive
) external onlyAllowedDeployers onlyAllowedYTokens(_yToken)
    returns (address deployedPool) {
    ...
    IERC20(_xToken).transferFrom(_msgSender(), address(deployedPool),
    _xAdd);
    ALTBCPool(deployedPool).initializePool(_msgSender(), _xAdd, _wInactive);
}
```

The problem here is that due to the use of the unsafe `transferFrom()`, if the xToken does not revert but returns false when the transfer fails, this causes the protocol to still use the `_xAdd` as the `initialLiq` when the token transfer fails, resulting in the deployer deploying the pool without having to provide initial liquidity.

```
function initializePool(address deployer, uint256 initialLiq,
    uint256 ___wInactive) external onlyOwner initializer {
    _w = int(initialLiq).toPackedFloat(POOL_NATIVE_DECIMALS_NEGATIVE);
    packedFloat __wInactive
    = int(___wInactive).toPackedFloat(POOL_NATIVE_DECIMALS_NEGATIVE);
    tbc.xMax = tbc.xMin.add(_w);
    packedFloat wActive = _w.sub(__wInactive);
    checkInactiveLiquidity(wActive, __wInactive);

    x = tbc.xMin;
    _updateParameters();

    packedFloat D0 = tbc.calculateDn(x);
    packedFloat initialRJ = D0.div((_w.sub(__wInactive)));
    ILPToken(lpToken).mintTokenAndUpdate(deployer, __wInactive,
    type(int256).max.toPackedFloat(0));
    emit PositionMinted(inactiveLpId, _msgSender(), true);
    emit LiquidityDeposited(deployer, inactiveLpId, ___wInactive, 0);

    ILPToken(lpToken).mintTokenAndUpdate(deployer, wActive, initialRJ);
    emit PositionMinted(activeLpId, _msgSender(), false);
    emit LiquidityDeposited(deployer, activeLpId, initialLiq - ___wInactive,
    0);
    _emitCurveState();
    _transferOwnership(deployer);
}
```

Recommendations:

It is recommended to use `safeTransferFrom()` to ensure successful token transfers.

Forte: Resolved with `@79ea050bcf ...`

Zenith: Verified.

4.2 Low Risk

A total of 1 low risk findings were identified.

[L-1] LPToken needs to disable `renounceOwnership()`

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

Target

- LPToken.sol#L19-L20

Description:

The change separates the LPToken and inherits the Ownable2Step contract, similar to FactoryBase, which needs to disable `renounceOwnership()` in order to avoid owner role renouncement that disables critical functions.

Recommendations:

It is recommended that LPToken disable `renounceOwnership()`.

Forte: Resolved with `@9057b9d6d4 ...`

Zenith: Verified.