QuillAudits

Audit Report
February, 2022

For

Amplify

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Scope of the Audit

The scope of this audit was to analyze and document the Amplify smart contracts codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis
Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis
Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis, Theo.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	3	13	0
Closed	0	1	13	0

Introduction

During the period of **December 26, 2021, to January 25, 2022** - QuillAudits Team performed a security audit for **Amplify** smart contracts.

The code for the audit was taken from the following official repo of Amplify: https://github.com/amplify-labs/contracts/tree/main/protocol/contracts

V	Date	Commit hash
1	January	d8af5f11f3b6ab59d09a56ebc229012900dc1c
2	January	c935b39804f5cdaaaf7e6926b86c07488f148671

Issues Found – Code Review / Manual Testing

A.Contract - Asset

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

A.1 Missing value verification

```
Line 23:
    function tokenizeAsset(
        string memory tokenHash,
        string memory tokenRating,
        uint256 value,
        uint256 maturity,
        string memory tokenURI
    ) external returns (uint256) {
        _tokenIds.increment();

        uint256 newAssetId = _tokenIds.current();
        _mint(msg.sender, newAssetId);

        _tokens[newAssetId] = Token(
            value,
            maturity,
            riskModel.getInterestRate(tokenRating),
            riskModel.getAdvanceRate(tokenRating),
            tokenRating,
            tokenHash,
            false
        );
```

```
Line 75:
    function addRiskItem(string memory rating, uint256 interestRate,
    uint256 advanceRate) external onlyOwner {
        riskModel.set(rating, interestRate, advanceRate);
    }

    function updateRiskItem(string memory rating, uint256 interestRate,
    uint256 advanceRate) external onlyOwner {
        riskModel.set(rating, interestRate, advanceRate);
    }
```

Description
Certain functions lack a safety check in the value, the values that are coming from the arguments should be verified, otherwise, the contract's functionality might get hurt.

Remediation
It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Closed

Fixed
The Amplify team has fixed the issue by adding require statements to verify the values provided from the arguments.

A.2 Floating Pragma

```
Line 3:
pragma solidity ^0.8.0;
```

Description
The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status: Closed

Fixed
The Amplify team has solved the issue by fixing the pragma version to 0.8.4.

B.Contract - AssetStorage

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

C.Contract - RiskModel

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

D.Contract - Controller

High severity issues

No issues were found.

Medium severity issues

D.1 For Loop Over Dynamic Array

```
Line 280:
for(uint8 i=0; i < borrowerPools[borrower].length; i++) {
            address pool = borrowerPools[borrower][i];
            pools[pool].isActive = false;
        }
```

Description

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocks and thus multiple transactions.

Status: Closed

Fixed

The Amplify Team has fixed the issue by limiting the amount of created pools.

Low severity issues

D.2 Missing address verification

```
Line 247:
    function whitelistLender(address _lender, address _pool) external returns (uint256) {
        Application storage application = poolApplicationsByLender[_pool][_lender];
        require(borrower == msg.sender, toString(Error.INVALID_OWNER));
```

```
Line 273:
function blacklistBorrower(address borrower) external onlyOwner returns (uint256) {
        Borrower storage _borrower = borrowers[borrower];

        require(_borrower.created, toString(Error.BORROWER_NOT_CREATED));
        require(_borrower.whitelisted, toString(Error.BORROWER_NOT_WHITELISTED));

        _borrower.whitelisted = false;
        for(uint8 i=0; i < borrowerPools[borrower].length; i++) {
            address pool = borrowerPools[borrower][i];
            pools[pool].isActive = false;
        }
        emit BorrowerBlacklisted(borrower);
        return uint256(Error.NO_ERROR);
    }
```

```
Line 289:
    function blacklistLender(address _lender) external returns (uint256) {
        require(borrowerWhitelists[msg.sender][_lender],
toString(Error.LENDER_NOT_WHITELISTED));

        borrowerWhitelists[msg.sender][_lender] = false;

Line 312:
    function addStableCoin(address stableCoin) onlyOwner external {
        require(_stableCoins.insert(stableCoin));
    }

    function removeStableCoin(address stableCoin) onlyOwner external {
        require(_stableCoins.remove(stableCoin));
    }

    function containsStableCoin(address stableCoin) public view returns (bool)
{       return _stableCoins.contains(stableCoin);
    }
```

Description
Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation
It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Closed

Fixed
The Amplify Team has fixed the issue by verifying the addresses provided in the arguments using a modifier.

D.3 Floating Pragma

```
Line 3:
pragma solidity ^0.8.0;
```

Description
The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status: Closed

Fixed
The Amplify team has solved the issue by fixing the pragma version to 0.8.4.

E.Contract - ControllerStorage

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

F.Contract - Rewards

High severity issues

No issues were found.

Medium severity issues

F.1 For Loop Over Dynamic Array

```
Line 26:
function getTotalBorrowReward(address account) external view returns (uint256) {
        uint256 totalAmount;
        for(uint256 i=0; i< rewardPools.length; i++) {
            totalAmount += this.getBorrowReward(account, rewardPools[i]);
        }
        return totalAmount;
    }
```

```
Line 39:
    function getTotalSupplyReward(address account) external view returns (uint256) {
        uint256 totalAmount;
        for(uint256 i=0; i< rewardPools.length; i++) {
            totalAmount += this.getSupplyReward(account, rewardPools[i]);
        }
        return totalAmount;
    }
Line 63:
    function claimAMPT(address[] memory holders, address[] memory poolsList, bool
borrowers, bool suppliers) public {
        for (uint8 i = 0; i < poolsList.length; i++) {
            address pool = poolsList[i];
            if (borrowers == true) {
                updateBorrowIndexInternal(pool);
                for (uint8 j = 0; j < holders.length; j++) {
                    distributeBorrowerTokens(pool, holders[j]);
                    borrowerState[holders[j]][pool].accrued = grantRewardInternal(holders[j],
borrowerState[holders[j]][pool].accrued);
                }
            }
            if (suppliers == true) {
                updateSupplyIndexInternal(pool);
                for (uint8 j = 0; j < holders.length; j++) {
                    distributeSupplierTokens(pool, holders[j]);
                    supplierState[holders[j]][pool].accrued = grantRewardInternal(holders[j],
supplierState[holders[j]][pool].accrued);
                }
            }
        }
    }
```

Description

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocks and thus multiple transactions.

Status: Acknowledged

Acknowledged

The Amplify team has acknowledged the risk since these methods are getters and won't affect the logic of the contract.

Low severity issues

F.2 Floating Pragma

```
Line 3:
pragma solidity ^0.8.0;
```

Description

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation

Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status: Closed

Fixed

The Amplify team has solved the issue by fixing the pragma version to 0.8.4.

G.Contract - StableCoin

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

G.1 Missing address verification

```
Line 12:
  function insert(Data storage self, address stableCoin) public returns (bool) {
        if (self.flags[stableCoin]) {
            return false;
        }

        self.flags[stableCoin] = true;
        self.addresses.push(stableCoin);
        self.addressIndex[stableCoin] = self.id;
        self.id++;
        return true;
    }
```

Description
Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation
It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Acknowledged

Acknowledged
The Amplify team has acknowledged the risk since the StableCoin contract it is used as a library by the Controller contract which has address verifications added.

G.2 Floating Pragma

```
Line 3:
pragma solidity ^0.8.0;
```

Description

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation

Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status: Closed

Fixed

The Amplify team has solved the issue by fixing the pragma version to 0.8.4.

H.Contract - ERC20

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

H.1 Approve Race

```
Line 62:
    function approve(address spender, uint amount) public virtual override returns (bool) {
        _approve(msg.sender, spender, amount);
        return true;
    }
```

Description

The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Remediation

Use increaseAllowance and decreaseAllowance function to modify the allowance value instead of overriding it using the approve function.

Status: Closed

Fixed

The Amplify team has added the increaseAllowance and decreaseAllowance to solve the issue.

H.2 Floating Pragma

```
Line 3:
pragma solidity ^0.8.0;
```

Description

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation

Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status: Closed

Fixed

The Amplify team has solved the issue by fixing the pragma version to 0.8.4.

I.Contract - ERC20Burnable

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

J.Contract - ERC20Mintable

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

K.Contract - ERC721

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

L.Contract - ERC721URIStorage

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

M.Contract - IERC721

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

N.Contract - AMPTChild

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

O.Contract - VotingEscrow

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

O.1 Missing Address Verification

```
Line 452:
    function _delegate(address delegator, address delegatee) internal {
        Balance storage _sourceBalance = _operationBalances[delegator];
        Balance memory _oldSourceBalance = _sourceBalance;

        require(_sourceBalance.amount > 0, "No existing lock found");

        Balance storage _destinationBalance = _operationBalances[delegatee];
        Balance memory _oldDestinationBalance = _destinationBalance;
        delegates[delegator] = delegatee;
        emit DelegateChanged(delegator, delegatee);

        _sourceBalance.amount = 0;
        /// @dev The balance.end should be intact for the future deposits
        _checkpoint(delegator, _oldSourceBalance, _sourceBalance);

        _destinationBalance.amount += _oldSourceBalance.amount;
        if(_oldDestinationBalance.end == 0) {
            _destinationBalance.end = _oldSourceBalance.end;
        }
        _checkpoint(delegatee, _oldDestinationBalance, _destinationBalance);
    }
```

Description
Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation
It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Acknowledged

Acknowledged
The Amplify team has acknowledged the risk.

O.2 Missing Address Verification

```
Line 607:
    function getBlockTimestamp() public virtual view returns (uint256) {
        return block.timestamp;
    }
```

Description
Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all that is guaranteed is that it is higher than the timestamp of the previous block.

Remediation
You can use an Oracle to get the exact time or verify if a delay of 900 seconds won’t destroy the logic of the staking contract.

Status: Acknowledged

Acknowledged
The Amplify team has acknowledged the risk.

O.3 Usage of Inline Assembly

```
Line 611:
    function getChainId() internal view returns (uint256) {
        uint256 chainId;
        assembly { chainId := chainid() }
        return chainId;
    }
```

Description

Inline assembly is a way to access the EVM at a low level. This discards several important safety features in Solidity.

Remediation

When possible, do not use inline assembly because it is a way to access the EVM at a low level. An attacker could bypass many important safety features of Solidity.

Status: Acknowledged

Acknowledged

The Amplify team has acknowledged the risk.

P.Contract - InterestRateModel

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

Q.Contract - InterestRateModel

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

Q.1 Missing Value Verification

```
Line 20:
    constructor(uint256 _blockPerYear) {
        blocksPerYear = _blockPerYear;
        predefinedStages();
    }
```

Description

Certain functions lack a safety check in the value, the values that are coming from the arguments should be verified, otherwise, the contract's functionality might get hurt.

Remediation

It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Closed

Fixed

The Amplify team has fixed the issue by adding a require statement to verify the value coming from the arguments.

Q.2 Floating Pragma

```
Line 3:
pragma solidity ^0.8.0;
```

Description

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation

Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status: Closed

Fixed

The Amplify team has solved the issue by fixing the pragma version to 0.8.4.

R.Contract - Borrower

High severity issues

No issues were found.

Medium severity issues

R.1 For Loop Over Dynamic Array

```
Line 56:
    function totalPrincipal() public virtual view returns (uint256) {
        uint256 total = 0;
        for (uint8 i = 0; i < creditLines.length; i++) {
            total += creditLines[i].principal;
        }
        return total;
    }

    function totalInterestRate() public virtual view returns (uint256) {
        uint256 total = 0;
        for (uint8 i = 0; i < creditLines.length; i++) {
            total += creditLines[i].interestRate;
        }
        if (total != 0){
            return total / creditLines.length;
        }
        return total;
    }

    /** @dev used by rewards contract */
    function getBorrowerTotalPrincipal(address _borrower) external view returns (uint256)
{
        uint256 balance;

        for(uint8 i=0; i < loansIdsByAddress[_borrower].length; i++) {
            uint256 loanId = loansIdsByAddress[_borrower][i];

            uint256 principal = creditLines[loanId].principal;
            bool penaltyStarted = penaltyInfo[loanId].isOpened;
            balance += penaltyStarted ? 0 : principal;
        }
        return balance;
    }
```

Description

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Status: Acknowledged

Acknowledged

The Amplify team has acknowledged the risk since it will affect only the pool.

Low severity issues

R.2 Missing Address Verification

```
Line 107:
    function createCreditLineInternal(address borrower, uint256 tokenId, uint256
borrowCap, uint256 interestRate, uint256 maturity) internal returns (uint256) {
        require(lockedAssetsIds[tokenId] == false,
toString(Error.LOAN_ASSET_ALREADY_USED));
        uint256 loanId = _loanIds.current();
        _loanIds;

        lockedAssetsIds[tokenId] = true;
        loansIdsByAddress[borrower].push(loanId);

        creditLines.push(CreditLine({
            borrower: borrower,
            borrowCap: borrowCap,
            borrowIndex: mantissaOne,
            principal: 0,
            lockedAsset: tokenId,
            interestRate: interestRate,
            accrualBlockNumber: getBlockNumber(),
            isClosed: false
        }));

        penaltyInfo[loanId] = PenaltyInfo({
            maturity: maturity,
            index: mantissaOne,
            timestamp: maturity + 30 days,
            isOpened: false
        });

        emit CreditLineOpened(loanId, tokenId, borrower, borrowCap, maturity,
interestRate);

        _loanIds.increment();
        return uint256(Error.NO_ERROR);
    }
```

```
Line 211:
    function repayInternal(uint256 loanId, address payer, address borrower, uint256
amount) internal onlyIfActive(loanId, borrower) nonReentrant returns (uint256) {
        uint256 allowed = repayAllowed(address(this), payer, borrower, amount);
        require(allowed == 0, toString(Error.CONTROLLER_REPAY_REJECTION));

        CreditLine storage creditLine = creditLines[loanId];
        PenaltyInfo storage _penaltyInfo = penaltyInfo[loanId];
        RepayLocalVars memory vars;

        vars.currentBorrowBalance = borrowBalanceSnapshot(loanId);
        (vars.penaltyIndex, vars.penaltyAmount) = getPenaltyIndexAndFee(loanId);
```

Description
Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation
It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Acknowledged

Acknowledged
The Amplify team has acknowledged the risk since it will affect only the pool.

R.3 Missing Value Verification

```
Line 107:
    function createCreditLineInternal(address borrower, uint256 tokenId, uint256
borrowCap, uint256 interestRate, uint256 maturity) internal returns (uint256) {
        require(lockedAssetsIds[tokenId] == false,
toString(Error.LOAN_ASSET_ALREADY_USED));
        uint256 loanId = _loanIds.current();
        _loanIds;

        lockedAssetsIds[tokenId] = true;
        loansIdsByAddress[borrower].push(loanId);

        creditLines.push(CreditLine({
            borrower: borrower,
            borrowCap: borrowCap,
            borrowIndex: mantissaOne,
            principal: 0,
            lockedAsset: tokenId,
            interestRate: interestRate,
            accrualBlockNumber: getBlockNumber(),
            isClosed: false
        }));

        penaltyInfo[loanId] = PenaltyInfo({
            maturity: maturity,
            index: mantissaOne,
            timestamp: maturity + 30 days,
            isOpened: false
        });


        emit CreditLineOpened(loanId, tokenId, borrower, borrowCap, maturity,
interestRate);

        _loanIds.increment();
        return uint256(Error.NO_ERROR);
    }
```

```
Line 174:
    function borrowInternal(uint256 loanId, address borrower, uint256 amount) internal
nonReentrant onlyIfActive(loanId, borrower) returns (uint256) {
        uint256 allowed = borrowAllowed(address(this), borrower, amount);
        require(allowed == 0, ErrorReporter.uint2str(allowed));

        CreditLine storage creditLine = creditLines[loanId];
        BorrowLocalVars memory vars;

        vars.currentTimestamp = getBlockTimestamp();
        require(vars.currentTimestamp < penaltyInfo[loanId].maturity,
toString(Error.LOAN_IS_OVERDUE));
```

```
Line 211:
    function repayInternal(uint256 loanId, address payer, address borrower, uint256
amount) internal onlyIfActive(loanId, borrower) nonReentrant returns (uint256) {
        uint256 allowed = repayAllowed(address(this), payer, borrower, amount);
        require(allowed == 0, toString(Error.CONTROLLER_REPAY_REJECTION));

        CreditLine storage creditLine = creditLines[loanId];
        PenaltyInfo storage _penaltyInfo = penaltyInfo[loanId];
        RepayLocalVars memory vars;

        vars.currentBorrowBalance = borrowBalanceSnapshot(loanId);
        (vars.penaltyIndex, vars.penaltyAmount) = getPenaltyIndexAndFee(loanId);
```

Description
Certain functions lack a safety check in the value, the values that are coming from the arguments should be verified, otherwise, the contract's functionality might get hurt.

Remediation
It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Acknowledged

Acknowledged
The Amplify team has acknowledged the risk.

R.4 Floating Pragma

```
Line 3:
pragma solidity ^0.8.0;
```

Description

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation

Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status: Closed

Fixed

The Amplify team has solved the issue by fixing the pragma version to 0.8.4.

S.Contract - Lender

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

T.Contract - Pool

High severity issues

No issues were found.

Medium severity issues

T.1 For Loop Over Dynamic Array

```
Line 157:
    function getTotalBorrowBalance() public virtual override(Lendable, Borrowable) view
returns (uint256) {
        uint256 total;
        for (uint8 i = 0; i < creditLines.length; i++) {
            total += borrowBalanceSnapshot(i);
        }
        return total;
    }
```

```
Line 229:
        for(uint8 i=0; i < _gracePeriod.length; i++) {
            uint256 _start = _gracePeriod[i].start * day + _penaltyInfo.maturity;
            uint256 _end = _gracePeriod[i].end * day + _penaltyInfo.maturity;

            if (vars.timestamp >= _start) {
                if(vars.timestamp > _end) {
                    vars.daysDelta = _calculateDaysDelta(_end, vars.accrualTimestamp, _start,
day);
                } else {
                    vars.daysDelta = _calculateDaysDelta(vars.timestamp,
vars.accrualTimestamp, _start, day);
                }

                vars.penaltyIndex = calculatePenaltyIndexPerPeriod(_gracePeriod[i].fee,
vars.interestBlocksPerYear, vars.daysDelta, vars.penaltyIndex);
                (vars.mathErr, vars.fee) = mulScalarTruncateAddUInt(Exp({mantissa:
vars.penaltyIndex }), vars.principal, vars.fee);
                ErrorReporter.check((uint256(vars.mathErr)));
            }
        }
```

Description
When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation
Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Status: Acknowledged

Acknowledged
The Amplify team has acknowledged the risk.

Low severity issues

T.2 Missing Address Verification

```
Line 37:
    function _initialize(address _admin, address _stableCoin, string memory _name,
uint256 _minDeposit, Access _access) internal nonReentrant {
        isInitialized = true;

        name = _name;
        minDeposit = _minDeposit;
        access = uint8(_access);

        // Set the admin address
        owner = _admin;

        // set the controller
        controller = ControllerInterface(msg.sender);

        // Set the stable coin contract
        stableCoin = IERC20Metadata(_stableCoin);

        lpToken = new PoolToken("PoolToken", stableCoin.symbol());
    }
```

Description
Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation
It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Acknowledged

Acknowledged
The Amplify team has acknowledged the risk.

T.3 Missing Value Verification

```
Line 37:
    function _initialize(address _admin, address _stableCoin, string memory _name,
uint256 _minDeposit, Access _access) internal nonReentrant {
        isInitialized = true;
        name = _name;
        minDeposit = _minDeposit;
        access = uint8(_access);
        // Set the admin address
        owner = _admin;
        // set the controller
        controller = ControllerInterface(msg.sender);
        // Set the stable coin contract
        stableCoin = IERC20Metadata(_stableCoin);

        lpToken = new PoolToken("PoolToken", stableCoin.symbol());
    }
```

Description

Certain functions lack a safety check in the value, the values that are coming from the arguments should be verified, otherwise, the contract's functionality might get hurt.

Remediation

It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Acknowledged

Acknowledged

The Amplify team has acknowledged the risk.

T.4 Usage Of block.timestamp

```
Line 325:
    function getBlockTimestamp() public virtual view returns (uint256) {
        return block.timestamp;
    }
```

Description

Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all that is guaranteed is that it is higher than the timestamp of the previous block.

Remediation

You can use an Oracle to get the exact time or verify if a delay of 900 seconds won't destroy the logic of the staking contract.

Status: Acknowledged

Acknowledged

The Amplify team has acknowledged the risk.

T.5 Floating Pragma

```
Line 3:
pragma solidity ^0.8.0;
```

Description

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation

Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status: Closed

Fixed

The Amplify team has solved the issue.

U.Contract - PoolToken

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

U.1 Approve Race

```
Line 7:
   contract PoolToken is ERC20Mintable {
```

Description

The standard ERC20 implementation contains a widely-known racing condition in it approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Remediation

Use increaseAllowance and decreaseAllowance function to modify the allowance value instead of overriding it using the approve function.

Status: Acknowledged

Acknowledged

The Amplify team has acknowledged the risk.

U.2 Floating Pragma

```
Line 3:
pragma solidity ^0.8.0;
```

Description

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation

Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status: Closed

Fixed

The Amplify team has fixed the issue.

V.Contract - LossProvisionPool

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

Functional Testing

Function name	Technical Result	Logical Result	Overall Result	Comment
				- Pool.sol indicates controller must follow ControllerInteraface, in Controller.sol interface seems not to be applied
				- functions amptToken(), provisionPool(), interestRateModel(), assetsFactor() in ControllerInterface.sol seem not to be used or implemented in implemented Controller.sol ?
				Notes Whitepaper(The protocol is implemented as a set of persistent, non-upgradable smart contracts -will upgrading be allowed for ?)
				Notes most files use(unlocked pragma ^0.8.0; multiple version rest ^0.8.0 but Governance and Vesting use 0.8.4;
				Maybe Natspec needs @title @param @return etc to ensure full NatSpec compliance for all the files like the commenting style in Governance/AMPT.sol
				Contracts in total may be very large in bytes size making it a very expensive project to deploy; e.g over 20KB => may need optimizations or simplifications
Controller Folder				
ControllerStorage.sol	PASS	PASS		Potential to save space in struct Application by moving created and whitelisted above depositAmount
Stablecoin.sol				
all functions	PASS	PASS		Is it not better to use external vs public for library functions?
Rewards.sol				
getTotalBorrowRewards	PASS	PASS		has a loop with array that can grow without bound that can be costly;

getBorrowReward	PASS	PASS		
getTotalSupplyReward	PASS	PASS?		has a loop with array that can grow without bound that can be costly;
getSupplyReward	PASS	PASS		
claimAMPT(holder)	PASS	PASS		
claimAMPT(holder, poolsList)	PASS	PASS		
claimAMPT(holder, poolsList, borrowers, suppliers)	PASS	PASS		has a loop with array that can grow without bound that can be costly ; if(borrowers==true) and for suppliers can just be if(borrowers),
updateBorrowIndexInternal	PASS	PASS		
getNewBorrowIndex	PASS	PASS		
updateSupplyIndexInternal	PASS	PASS		
getNewSupplyIndex	PASS	PASS		
distributeBorrowerTokens	PASS	PASS		
getBorrowerAccruedAmount	PASS	PASS		
distributeSupplierTokens	PASS	PASS		
getSupplierAccruedAmount	PASS	PASS		
getBorrowerTotalPrincipal	PASS	PASS		
getSupplierBalance	PASS	PASS		
getPoolInfo	PASS	PASS		

grantRewardInternal	PASS	PASS		
getBlockNumber	PASS	PASS		
Controller.sol				
constructor	PASS	PASS		
_deployPoolLibrary	PASS	PASS		
getBorrowerPools	PASS	PASS		
transferAMPTDeposit	PASS	PASS		
submitBorrower	PASS	PASS		
requestPoolWhitelist	PASS	PASS		
withdrawApplicationDeposit	PASS	PASS		
createPool	PASS	PASS		
getPoolUtilizationRate	PASS	PASS		
getPoolAPY	PASS	PASS		
getTotalSupplyBalance	PASS	PASS		Looping over dynamic array that can grow without bounds
_setProvisionPool	PASS	PASS		
_setInterestRateModel	PASS	PASS		
_setAssetsFactory	PASS	PASS		
_setAmptContract	PASS	PASS		
_setAmptSpeed	PASS	PASS		

_setAmptDepositAmount	PASS	PASS		
transferFunds	PASS	PASS		
whitelistBorrower	PASS	PASS		Zero address check borrower to avoid unnecessary read from storage gas costs
whitelistLender	PASS	PASS?		Zero address check borrower to avoid unnecessary read from storage gas costs
blacklistBorrower	PASS	PASS		Zero address check borrower to avoid unnecessary read from storage gas costs
blacklistLender	PASS	PASS		
updateBorrowerInfo	PASS	PASS?		
addStableCoin	PASS	PASS		Maybe an event should be emitted
removeStableCoin	PASS	PASS		Maybe an event should be emitted
containsStableCoin	PASS	PASS		
getStableCoins	PASS	PASS		
lendAllowed	PASS	PASS		
redeemAllowed	PASS	PASS		
borrowAllowed	PASS	PASS		
repayAllowed	PASS	PASS		
createCreditLineAllowed	PASS	PASS		
calculateBorrowInterestRate	PASS	PASS		
transferAMPTDeposit	PASS	PASS		
_setBorrowerDebtCeiling	PASS	PASS		

_setBorrowerRating	PASS	PASS		
grantRewardInternal	PASS	PASS		
getBorrowerTotalPrincipal	PASS	PASS		
getSupplierBalance	PASS	PASS		
getPoolInfo	PASS	PASS		
getBlockNumber	PASS	PASS		
getBlockTimestamp	PASS	PASS		
Pool				
PoolToken.sol	PASS	PASS		
createPoolTokenSymbol	PASS	PASS		
Borrower.sol	PASS	PASS		
				Potential to save space in struct CreditLine by moving bool isClosed above borrowCap
				Potential to save space in struct PenaltyInfo by reducing index size and moving it next to bool isOpened
_isActive	PASS	PASS		
totalPrincipal	PASS	PASS		looping cost over dynamic array creditLines which can grow and cause DOS
totalInterestRate	PASS	PASS		looping cost over dynamic array creditLines which can grow and cause DOS
getBorrowerTotalPrincipal	PASS	PASS		
getBorrowerBalance	PASS	PASS		
borrowerSnapshot	PASS	PASS		
getBorrowerLoans	PASS	PASS		

createCreditLineInternal	PASS	PASS		
closeCreditLineInternal	PASS	PASS		no check for loanId can lead to unnecessary looking up storage
unlockAssetInternal	PASS	PASS		no check for loanId can lead to unnecessary looking up storage
borrowInternal	PASS	PASS		
repayInternal	PASS	PASS		
borrowBalanceSnapshot	PASS	PASS		no check for loanId can lead to unnecessary looking up storage
Lender.sol				
lendInternal	PASS	PASS		
redeemInternal	PASS	PASS		
exchangeRate	PASS	PASS		
exchangRateInternal	PASS	PASS		
balanceOf	PASS	PASS		
balanceOfUnderlying	PASS	PASS		
totalSupply	PASS	PASS		
Pool.sol				
_initialize	PASS	PASS		
initialize	PASS	PASS		
changeAccess	PASS	PASS		
_lend	PASS	PASS		
lend	PASS	PASS		
redeem	PASS	PASS		
redeemUnderlying	PASS	PASS		
_transferTokens	PASS	PASS		

getCash	PASS	PASS		
lendAllowed	PASS	PASS		
redeemAllowed	PASS	PASS		
createCreditLine	PASS	PASS		
closeCreditLine	PASS	PASS		
redeemAsset	PASS	PASS		
unlockAsset	PASS	PASS		
borrow	PASS	PASS		
repay	PASS	PASS		
repayBehalf	PASS	PASS		
getBorrowIndex	PASS	PASS		
getPenaltyIndexAndFee	PASS	PASS		
_calculateDaysDelta	PASS	PASS		
calculatePenaltyIndexPerPeriod	PASS	PASS		
_transferTokensOnBorrow	PASS	PASS		
_transferTokensOnRepay	PASS	PASS		
borrowAllowed	PASS	PASS		
repayAllowed	PASS	PASS		
getBlockNumber	PASS	PASS		
getBlockTimestamp	PASS	PASS		

Results

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines. Many issues were discovered during the initial audit; the majority of them are fixed.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **Amplify** Contracts. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **Amplify** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
February, 2022

For

Amplify

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com