SALUS SECURITY

Mar 2024

CODE SECURITY ASSESSMENT

PAC FINANCE

Overview

Project Summary

- Name: Pac Finance
- Platform: Blast
- Language: Solidity
- Repository:
  - https://github.com/Pac-Fi/pac-finance
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Pac Finance
Version	v3
Type	Solidity
Dates	Apr 01 2024
Logs	Mar 27 2024; Mar 31 2024; Apr 01 2024

Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	1
Total Low-Severity issues	3
Total informational issues	1
Total	7

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction **4**

1.1 About SALUS 4

1.2 Audit Breakdown 4

1.3 Disclaimer 4

Findings **5**

2.1 Summary of Findings 5

2.2 Notable Findings 6

1. Stable rate mode borrowing should be disabled 6
2. Funds can be drained via one empty market 7
3. Centralization risk 9
4. Users may fail to withdraw the expected number of tokens through PacPoolWrapper 10
5. Lack of validation for depositAsset 12
6. CToken’s price might be out-of-date 14

2.3 Informational Findings 15

7. Gas optimization suggestions 15

Appendix **16**

Appendix 1 - Files in Scope 16

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Stable rate mode borrowing should be disabled	High	Business Logic	Resolved
2	Funds can be drained via one empty market	High	Numerics	Resolved
3	Centralization risk	Medium	Centralization	Mitigated
4	Users may fail to withdraw the expected number of tokens through PacPoolWrapper	Low	Numerics	Acknowledged
5	Lack of validation for depositAsset	Low	Data Validation	Resolved
6	CToken’s price might be out-of-date	Low	Business Logic	Acknowledged
7	Gas optimization suggestions	Informational	Gas Optimization	Acknowledged

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Stable rate mode borrowing should be disabled

Severity: High	Category: Business Logic

Target:
- contracts/core-v3/contracts/protocol/tokenization/StableDebtToken.sol

Description

Similar to AAVE, users in Pac Finance can borrow in two ways, stable debt or variable debt. AAVE protocol has disabled stable debt tokens because of an unpublished vulnerability.

Referring to AAVE's proposal, stable debt tokens have been upgraded to prevent new mining of StableDebt.

contracts/core-v3/contracts/protocol/tokenization/StableDebtToken.sol:L150-L216

```
function mint(
    address user,
    address onBehalfOf,
    uint256 amount,
    uint256 rate
) external virtual override onlyPool returns (bool, uint256, uint256) {
    MintLocalVars memory vars;

    if (user != onBehalfOf) {
        _decreaseBorrowAllowance(onBehalfOf, user, amount);
    }
    ...
}
```

Recommendation

Consider updating the mint() function according to the AAVE's patch.

```
function mint(
  address,
  address,
  uint256,
  uint256
) external virtual override onlyPool returns (bool, uint256, uint256) {
  revert('STABLE_BORROWING_DEPRECATED');
}
```

Status

This issue has been resolved by the team with commit 301e46f.

2. Funds can be drained via one empty market

Severity: High	Category: Numerics

Target:

- contracts/core-v3/contracts/protocol/tokenization/base/ScaledBalanceTokenBase.sol
- contracts/core-v3/contracts/protocol/libraries/math/WadRayMath.sol

Description

As one lending protocol, Pac Finance needs to pay attention to the empty market issue, especially as one AAVE fork. Hackers can drain all funds via one empty market, refer to the HopeLend incident.

Attack vectors are as follows:

- Hacker, as the first depositor, mint some AToken;
- The hacker can manipulate AToken's price by repeatedly transferring tokens to the AToken contract and withdrawing;
- The hacker can then borrow assets from other asset markets using the inflated AToken.

contracts/core-v3/contracts/protocol/tokenization/base/ScaledBalanceTokenBase.sol:L112-L119

```
function _burnScaled(
       address user,
       address target,
       uint256 amount,
       uint256 index
) internal {
       uint256 amountScaled = amount.rayDiv(index);
       require(amountScaled != 0, Errors.INVALID_BURN_AMOUNT);
```

contracts/core-v3/contracts/protocol/libraries/math/WadRayMath.sol:L90-L102

```
function rayDiv(uint256 a, uint256 b) internal pure returns (uint256 c) {
    // to avoid overflow, a <= (type(uint256).max - halfB) / RAY
    assembly {
        if or(
            iszero(b),
            iszero(iszero(gt(a, div(sub(not(0), div(b, 2)), RAY))))
        ) {
            revert(0, 0)
        }

        c := div(add(mul(a, RAY), div(b, 2)), b)
    }
}
```

Recommendation

When deploying a new market, it is recommended to mint some ATokens as the first depositor to prevent the empty market attack.

Status

This issue has been resolved by the team. The project team would set both LTV and LiquidatonThreshold to 0 for new assets before minting aToken themselves.

3. Centralization risk

Severity: Medium	Category: Centralization

Target:

- contracts/NativeYieldDistribute.sol

Description

There is a privileged owner role in the NativeYieldDistribute contract. The owner of the NativeYieldDistribute contract can distribute and withdraw all funds in the smart contract.

Should the owner’s private key be compromised, an attacker could withdraw all yield distribution.

Since the privileged account is a plain EOA account, this can be worrisome and pose a risk to the other users.

contracts/NativeYieldDistribute.sol:L217-L228

```
function rescueToken(
    address token,
    address to,
    uint256 amount
) external onlyOwner {
    if (token == address(0)) {
        _safeTransferETH(to, amount);
    } else {
        IERC20(token).safeTransfer(to, amount);
    }
    emit RescueToken(token, to, amount);
}
```

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been mitigated by the team.

4. Users may fail to withdraw the expected number of tokens through PacPoolWrapper

Severity: Low	Category: Numerics

Target:
- contracts/PacPoolWrapper.sol
- contracts/core-v3/contracts/protocol/tokenization/base/ScaledBalanceTokenBase.sol

Description

When a user wants to withdraw ETH or ERC20 tokens through the PacPoolWrapper contract, aTokens will first be transferred from the user to the contract. The PacPoolWrapper will then call the withdraw() function of the AAVE pool contract with the amountToWithdraw parameter.

contracts/PacPoolWrapper.sol:L269-L290

```
function withdrawERC20(
        address asset,
        uint256 amount,
        address to
) external nonReentrant {
        IAToken aToken = IAToken(
        POOL.getReserveData(address(asset)).aTokenAddress
        );
        uint256 userBalance = aToken.balanceOf(msg.sender);
        uint256 amountToWithdraw = amount;

        // if amount is equal to uint(-1), the user wants to redeem everything
        if (amount == type(uint256).max) {
        amountToWithdraw = userBalance;
        }
        aToken.transferFrom(msg.sender, address(this), amountToWithdraw);

        uint256 gasBegin = gasleft();
        POOL.withdraw(asset, amountToWithdraw, to);
        uint256 gasEnd = gasleft();
        _addGasRefund(gasBegin - gasEnd, IGasRefund.RefundType.WITHDRAW);
}
```

The actual aToken transfer amount will be calculated based on the liquidityIndex. There could be a loss of precision due to the use of rayDiv(), resulting in tokens received by the PacPoolWrapper being smaller than amountToWithdraw.

contracts/core-v3/contracts/protocol/tokenization/AToken.sol:L284

```
function _transfer(
        address from,
        address to,
        uint256 amount,
        bool validate
) internal virtual {
       address underlyingAsset = _underlyingAsset;
        uint256 index = POOL.getReserveNormalizedIncome(underlyingAsset);
        ...
        super._transfer(from, to, amount, index);
```

contracts/core-v3/contracts/protocol/tokenization/base/ScaledBalanceTokenBase.sol:L166

```
function _transfer(
        address sender,
        address recipient,
        uint256 amount,
        uint256 index
) internal {
        ...
        super._transfer(sender, recipient, amount.rayDiv(index).toUint128());
```

Recommendation

Consider withdrawing based on the number of tokens received and informing users that the remaining tokens can also be withdrawn directly through the withdraw() function of the AAVE pool contract.

Status

This issue has been acknowledged by the team.

5. Lack of validation for depositAsset

Severity: Low	Category: Data Validation

Target:
- contracts/PacPoolWrapper.sol

Description

In the multiplierDeposit() function, users need to provide the address of the deposit asset.

contracts/PacPoolWrapper.sol:L392-L401

```
function multiplierDeposit(
        address asset,
        uint256 cashAmount,
        uint256 borrowAmount,
        address depositAsset,
        uint256 minDepositAmount,
        address[] calldata swapPath
) external payable nonReentrant {
        if (address(swapRouter) == address(0)) revert FeatureNotActive();
        if (asset == depositAsset) revert InvalidParam();
```

The actual deposit asset will be the last address element in the swapPath. However, there is no validation to ensure that the last element in the swapPath and the depositAsset are the same.

contracts/PacPoolWrapper.sol:L482-L533

```
function executeOperation(
        address asset,
        uint256 amount,
        uint256 premium,
        address initiator,
        bytes calldata params
) external returns (bool) {
        OperationType operationType = abi.decode(params, (OperationType));

        if (operationType == OperationType.Leverage) {
            ...
        } else if (operationType == OperationType.Multiplier) {
            ...
            address supplyAsset = localVars.swapPath[
               localVars.swapPath.length - 1
            ];
            _checkApprove(supplyAsset, address(POOL));
            POOL.supply(
               supplyAsset,
               amounts[amounts.length - 1],
               localVars.user,
               referralCode
            );
            ...
        }

        //will revert in lending pool
        return false;
}
```

Recommendation

Consider validating if the last element in the swapPath is the depositAsset.

Status

This issue has been resolved by the team with commit f01bbcc.

6. CToken’s price might be out-of-date

Severity: Low	Category: Business Logic

Target:
- contracts/ExchangeRateAssetPriceAdapter.sol

Description

In the ExchangeRateAssetPriceAdapter::latestAnswer() function, the contract will calculate cToken’s price based on the underlying token’s price and cToken’s exchange rate.

CToken’s exchange rate can only be updated via syncExchangeRate() by the owner, which might be out-of-date.

contracts/ExchangeRateAssetPriceAdapter.sol:L31-L34

```
function syncExchangeRate() external onlyOwner {
    cTokenExchangeRate = ICToken(cToken).exchangeRateStored().toInt256();
}

function latestAnswer() public view virtual override returns (int256) {
    int256 underlyingPrice = underlyingAssetOracle.latestAnswer();
    return underlyingPrice * cTokenExchangeRate / 1e18;
}
```

Recommendation

The owner needs to sync the exchange rate in time, e.g. by using off-chain bots, to ensure that the difference between the cTokenExchangeRate and the actual cTokenExchangeRate is acceptable.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

7. Gas optimization suggestions

Severity: Informational	Category: Gas Optimization
Target: - contracts/NativeYieldDistribute.sol	

Description

info.currentBalance will not change during the loop, consider using a local variable to cache.

contracts/NativeYieldDistribute.sol:L150-L163

```
for (
        uint256 index = info.pointSettleRound + 1;
        index < currentRound;
        index++
) {
        uint256 roundEndTime = roundSettleTime[index];
        uint256 roundDuration = roundEndTime - info.balanceUpdateTime;
        uint256 totalRoundPoint = info.settledPoint +
        info.currentBalance *
        roundDuration;
        ...
}
```

Recommendation

Consider using a local variable to cache the info.currentBalance.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit ca304a3:

File	SHA-1 hash
contracts/API3OracleWrapper.sol	4027e5bae214119291792ed592e4e7ad0920fda0
contracts/CLFixedPriceSynchronicityPriceAdapter.sol	034c6aaaa318bb42cae0fa4161592bee0810c2ed
contracts/ERC20OracleWrapper.sol	0cdb5216af81cbcd5605324f32029161c905e0e0
contracts/ExchangeRateAssetPriceAdapter.sol	262b728e8dc980f328f396571279e927e6371364
contracts/GasRefund.sol	719d51e70b7ced2eac0ca449867f6902c81f4b95
contracts/NativeYieldDistribute.sol	0bc71790f6e7ba78c3b801e2e4830e524fe33284
contracts/PacPoolWrapper.sol	951084271a815ebc985a89b2ea4fa649ae5cb47a
contracts/UniswapV2OracleWrapper.sol	53bfac579c4e44d929e8263579eb90ac2330b909
contracts/core-v3/contracts/protocol/pool/Pool.sol	2da69dafe4fc19c24a1f6878eb6cc9952a1f9481
contracts/core-v3/contracts/protocol/tokenization/AToken.sol	59f8fb6b23deba254da0316815711f3de6d60ddb
contracts/core-v3/contracts/protocol/tokenization/base/ScaledBalanceTokenBase.sol	96373dae64b7e5ce876dc0319efe03afbd9e9133
contracts/core-v3/contracts/protocol/tokenization/StableDebtToken.sol	9792e0a17bc8683acda6cf55d7eeb290c211d299
contracts/core-v3/contracts/protocol/tokenization/VariableDebtToken.sol	4bc1bb3f3f100fb4ca9d5fe52cfb7f8c59c9c298