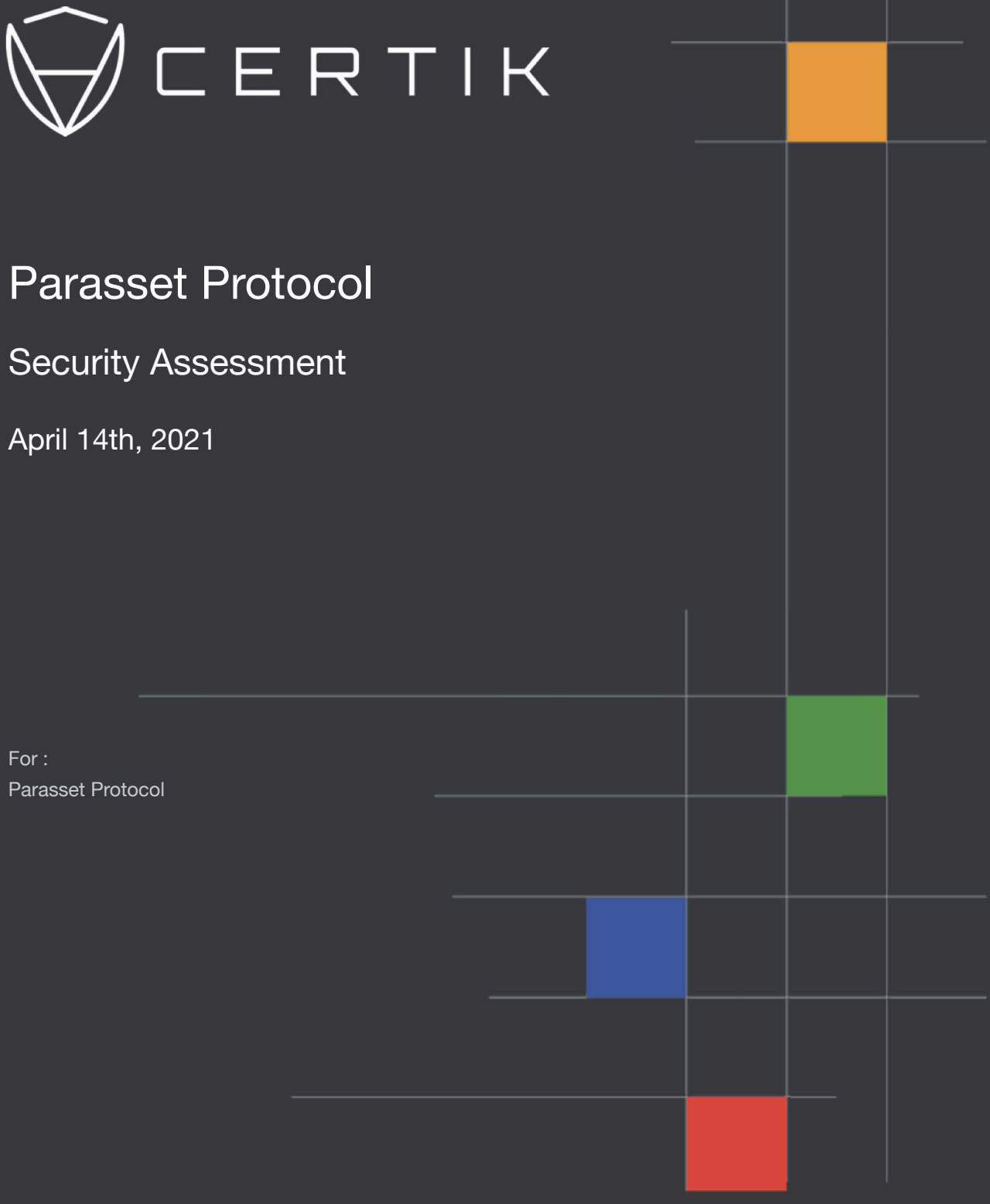


Parasset Protocol

Security Assessment

April 14th, 2021

For :
Parasset Protocol

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Parasset Protocol
Description	Decentralized Finance Protocol
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commits	b4c3be38dda757449c88687b8ad50007a3ea946e

Audit Summary

Delivery Date	April. 14th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Mar. 12, 2021 - April. 14, 2021

Vulnerability Summary

Total Issues	15
Total Critical	0
Total Major	1
Total Minor	6
Total Informational	5
Total Discussion	3

Executive Summary

This report has been prepared for Parasset Protocol smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

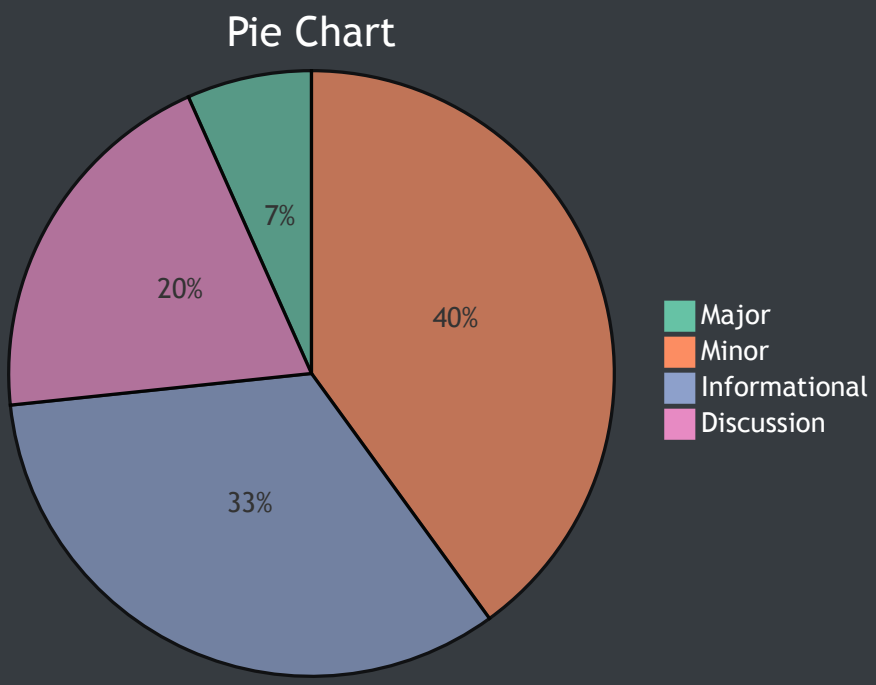
The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

File in Scope

ID	Contract	SHA256-Checksum
IP	InsurancePool.sol	e6c9318c5a06c91ce5f28270698bbc8c8a7db5cdc0d29542ca1c1cf00da35648
MP	MortgagePool.sol	3adab02730af37f37a5dc2ca1aa64bbc17a2f2e377fd5383343f1c9292bb61d3
PT	PToken.sol	d59224e701d8b419a17219966e21df71d3919c557a9bb13ad7a01b4e24ea99cd
PF	PTokenFactory.sol	74a25464fadbc44298ff9ca9c4dfaa912c4387acfc8172a67ab96a9ccfd7c950
PC	PriceController.sol	da78c5252999403f59ed2ce52428ffb48c17fc9c2ad3009c03d0de7dc9fc93e5

Findings



ID	Title	Type	Severity	Resolved
IP-01	Possible Re-entrancy	Language Specific	Minor	✓
IP-02	pToken Loss	Volatile Code	Minor	✓
IP-03	Ether Loss	Language Specific	Major	✓
IP-04	Non-zero Amount	Coding Style	Minor	✓
IP-05	Uncertain Code	Logical Issue	Discussion	✓
IP-06	Better Method Of pToken	Coding Style	Informational	✓
IP-07	Wrong Arguments	Logical Issue	Major	✓
IP-08	Replace Declaration	Gas Optimization	Informational	✓
MP-01	Bad Debts	Logical Issue	Discussion	
MP-02	Restrict Arguments	Coding Style	Informational	✓
MP-03	Redundant Codes	Dead Code	Informational	✓
MP-04	Fee Loss	Logical Issue	Minor	✓
MP-05	Ever-growing Ledger Array	Logical Issue	Discussion	✓
MP-06	Transfer Failure	Language Specific	Minor	✓
PC-01	Return Zero	Volatile Code	Informational	✓

IP-01: Possible Re-entrancy

Type	Severity	Location
Language Specific	🔵Minor	InsurancePool.sol L218,L236 L259,L299 ,L350, Mortgage.sol, L306,L356,L398,L438,L474,L509,L550

Description:

Transferring ETH to an external address may cause a re-entrancy attack.

In the `exchangePTokenToUnderlying()` function:

```
    function exchangePTokenToUnderlying(address pToken, uint256 amount) public whenActive {
        ......
        payEth(address(msg.sender), pTokenAmount);
        ......
        }
```

In the `redemptionIns()` function:

```
    function redemptionIns(address token, uint256 amount) public noStop {
        ......
        if (tokenBalance >= underlyingAmount) {
            payEth(address(msg.sender), underlyingAmount);
            } else {
                payEth(address(msg.sender), tokenBalance);
        ......
        }
```

Recommendation:

Using the ReentrancyGuard library, see :ReentrancyGuard.sol

An example revision is shown below:

```
    pragma solidity ^0.6.12;

    import "https://github.com/OpenZeppelin/openzeppelin-
    contracts/blob/master/contracts/security/ReentrancyGuard.sol";

    contract InsurancePool is ReentrancyGuard{
        ......
```

```
    function exchangePTokenToUnderlying(address pToken, uint256 amount) public whenActive
nonReentrant{
        ......
        payEth(address(msg.sender), pTokenAmount);
        ......
    }
    ......
    function redemptionIns(address token, uint256 amount) public noStop nonReentrant{
        ......
        if (tokenBalance >= underlyingAmount) {
            payEth(address(msg.sender), underlyingAmount);
        } else {
            payEth(address(msg.sender), tokenBalance);
        ......
    }
```

The same problem exists in MortgagePool smart contract.

Alleviation:

The development team heeded our advice and resolved this issue in commit

71bf95eb56af3bd21acfa685401e21bfccbe4eed

IP-02: pToken Loss

Type	Severity	Location
Volatile Code	Minor	InsurancePool.sol L223,L246

Description:

When the value of `amount` is too small, `pTokenAmount` will be equal to zero. In this case, user will lose funds.

In the `exchangePTokenToUnderlying()` function:

```
    function exchangePTokenToUnderlying(address pToken,
                                        uint256 amount) public whenActive {
        ......
          uint256 pTokenAmount = getDecimalConversion(pToken, amount.sub(fee),
    underlyingToken);
        if (underlyingToken != address(0x0)) {
          ERC20(underlyingToken).safeTransfer(address(msg.sender), pTokenAmount);
        } else {
          payEth(address(msg.sender), pTokenAmount);
        }
        ......
      }
    function exchangeUnderlyingToPToken(address token,
                                        uint256 amount) public payable whenActive {
        ......
          uint256 pTokenAmount = getDecimalConversion(token, amount.sub(fee), pToken);
          ......
      }
```

Examples:

Considering that `underlyingToken` is USDT, the value of `amount` must be greater than `1e12`.

Recommendation:

Add condition checking about `pTokenAmount`. An example revision is shown below:

```
    function exchangePTokenToUnderlying(address pToken,
                                        uint256 amount) public whenActive {
        ......
          uint256 pTokenAmount = getDecimalConversion(pToken, amount.sub(fee),
    underlyingToken);
          require(pTokenAmount > 0, "Log:InsurancePool:!amount");
        if (underlyingToken != address(0x0)) {
          ERC20(underlyingToken).safeTransfer(address(msg.sender), pTokenAmount);
```

```
        } else {
          payEth(address(msg.sender), pTokenAmount);
        }
        ......
      }
    function exchangeUnderlyingToPToken(address token,
                                        uint256 amount) public payable whenActive {
        ......
        uint256 pTokenAmount = getDecimalConversion(token, amount.sub(fee), pToken);
        require(pTokenAmount > 0, "Log:InsurancePool:!amount");
        ......
      }
```

Alleviation:

The development team heeded our advice and resolved this issue in commit

71bf95eb56af3bd21acfa685401e21bfccbe4eed

IP-03: Ether Loss

Type	Severity	Location
Language Specific	Major	InsurancePool.sol L218

Description:

The contract of pToken might be maliciously forged. Consider the following scenarios: the attacker creates a fake pToken, then he will obtain ETH.

In the `exchangePTokenToUnderlying()` function:

```
    function exchangePTokenToUnderlying(address pToken,
                                        uint256 amount) public whenActive {
          ......
          ERC20(pToken).safeTransferFrom(address(msg.sender), address(this), amount);
            // underlyingToken will equal to 0x00
            address underlyingToken = mortgagePool.getPTokenToUnderlying(pToken);
            uint256 pTokenAmount = getDecimalConversion(pToken, amount.sub(fee),
    underlyingToken);
          if (underlyingToken != address(0x0)) {
            ERC20(underlyingToken).safeTransfer(address(msg.sender), pTokenAmount);
          } else {
            payEth(address(msg.sender), pTokenAmount);
          }
          ......
        }
```

Recommendation:

Note that the variables of contracts are initialized to zero by default. One recommended fix is to associate the ETH address to a specific non-zero address. For instance, `address(uint256(-1))` .

Method 2: Using `getUnderlyingToPToken()` and `getPTokenToUnderlying()` to validate the pToken contract. An example revision is shown below:

```
    function exchangePTokenToUnderlying(address pToken,
                                        uint256 amount) public whenActive {
          ......
            address underlyingToken = mortgagePool.getPTokenToUnderlying(pToken);
            address Token = mortgagePool.getUnderlyingToPToken(pToken);
            require(underlyingToken != address(0x0) || Token != address(0x0),
    "Log:InsurancePool:!pToken");
            ......
        }
```

Alleviation:

The development team heeded our advice and resolved this issue in commit

71bf95eb56af3bd21acfa685401e21bfccbe4eed

IP-04: Non-zero Amount

Type	Severity	Location
Coding Style	Minor	InsurancePool.sol L218,L236,L259, L299

Description:

When the amount of subscribing or redeeming insurance by the user is zero, the function execution is meaningless.

These functions are as follows:

```
    function exchangePTokenToUnderlying(address pToken,
                                        uint256 amount) public whenActive {
          ......
        }
    function exchangeUnderlyingToPToken(address token,
                                        uint256 amount) public payable whenActive {
          ......
        }
    function subscribeIns(address token,
                          uint256 amount) public payable whenActive {
            ......
        }

    function redemptionIns(address token,
                           uint256 amount) public noStop {
            ......
        }
```

Recommendation:

Check `amount` first. An example revision is shown below:

```
    function exchangePTokenToUnderlying(address pToken,
                                        uint256 amount) public whenActive {
          require(amount > 0, "Log:InsurancePool:!amount");
            ......
        }
    function exchangeUnderlyingToPToken(address token,
                                        uint256 amount) public payable whenActive {
          require(amount > 0, "Log:InsurancePool:!amount");
            ......
        }
    function subscribeIns(address token,
                          uint256 amount) public payable whenActive {
          require(amount > 0, "Log:InsurancePool:!amount");
```

```
        ......
    }

    function redemptionIns(address token,
                           uint256 amount) public noStop {
        require(amount > 0, "Log:InsurancePool:!amount");
        ......
    }
```

Alleviation:

The development team heeded our advice and resolved this issue in commit

71bf95eb56af3bd21acfa685401e21bfccbe4eed

IP-05: Uncertain Code

Type	Severity	Location
Logical Issue	Discussion	InsurancePool.sol L281

Description:

It seems to be different from the formula provided in the technical white paper:

```
    function subscribeIns(address token,
                          uint256 amount) public payable whenActive {
            ......
            insAmount = getDecimalConversion(token, amount, pToken).mul(insTotal).div(allValue);

          ......
          // 增发份额
          issuance(token, insAmount, address(msg.sender));
          // 冻结保险份额
          frozenInfo.amount = frozenInfo.amount.add(insAmount);
          ......
        }
```

Alleviation:

After discussing with the developer team, we have confirmed that the issue has been resolved.

IP-06: Better Method Of pToken

Type	Severity	Location
Coding Style	Informational	InsurancePool.sol

Description:

It is recommended that the decimal settings of pToken and token be the same to avoid the problem of losing funds due to fractional conversion. For example, set pUSDT's decimal to `1e6`.

Alleviation:

This issue has been discussed and resolved.

IP-07: Wrong Arguments

Type	Severity	Location
Logical Issue	Major	InsurancePool.sol, L380

Description:

After `insNegative[token]` is reset to zero, negative assets cannot be eliminated:

```
    function _eliminate(address pToken, address token) private {
          PToken pErc20 = PToken(pToken);
          uint256 negative = insNegative[token];
          .......
              insNegative[token] = 0;
              pErc20.destroy(insNegative[token], address(this));
            }
          }
        }
```

Recommendation:

Use local variable `negative` instead of `insNegative[token]`. An example revision is shown below:

```
    function _eliminate(address pToken, address token) private {
          PToken pErc20 = PToken(pToken);
          uint256 negative = insNegative[token];
          .......
              insNegative[token] = 0;
              pErc20.destroy(negative, address(this));
            }
          }
        }
```

Alleviation:

The development team heeded our advice and resolved this issue in commit

71bf95eb56af3bd21acfa685401e21bfccbe4eed

IP-08: Replace Declaration

Type	Severity	Location
Gas Optimization	Informational	MortgagePool.sol InsurancePool.sol

Description:

The declaration of `public` functions that are never called by the contract should be declared `external` to save gas.

For example, some functions are as follows:

```
 function getGovernance() public view returns(address) {
        return governance;
    }

function getInsNegative(address token) public view returns(uint256) {
        return insNegative[token];
    }

function getTotalSupply(address token) public view returns(uint256) {
        return totalSupply[token];
    }

function getBalances(address token, address add) public view returns(uint256) {
        return balances[add][token];
    }
```

Recommendation:

Use the `external` attribute for functions never called from the contract.

Alleviation:

The development team heeded our advice and resolved this issue in commit

71bf95eb56af3bd21acfa685401e21bfccbe4eed

MP-01: Bad Debts

Type	Severity	Location
Logical Issue	Discussion	MortgagePool.sol

Description:

Since any user can liquidate any account when the conditions are met. Consider the following case: Alice deliberately "increases" her mortgage rate, and if the price of the mortgage asset falls to meet the liquidation criteria, then Alice liquidates her account and spends less pToken.

Given the plummeting price of mortgaged assets, for example (rare but possible), 20% of the original mortgage price, no one will redeem or liquidate the collaterals, resulting in bad debts. No countermeasure would be taken in the system to mitigate such risks.

Alleviation:

No alleviation.

MP-02: Restrict Arguments

Type	Severity	Location
Coding Style	Informational	MortgagePool.sol L356,L398,L438,L474

Description:

When the user carelessly enters an amount of zero, a fee will be lost:

```
    function supplement(address mortgageToken,
                        address pToken,
                        uint256 amount) public payable whenActive {
          ......
          if (parassetAssets > 0 && block.number > blockHeight && blockHeight != 0) {
                // 结算稳定费
                uint256 fee = getFee(parassetAssets, blockHeight, pLedger.rate);
                // 转入p资产
                ERC20(pToken).safeTransferFrom(address(msg.sender), address(insurancePool),
    fee);
                // 消除负账户
                insurancePool.eliminate(pToken, pTokenToUnderlying[pToken]);
                emit FeeValue(pToken, fee);
          }
          ......
        }
```

Recommendation:

Add condition checks about `amount`. An example revision is shown below:

```
    function supplement(address mortgageToken,
                        address pToken,
                        uint256 amount) public payable whenActive {
            require(amount > 0, "Log:MortgagePool:!amount");
          ......
          if (parassetAssets > 0 && block.number > blockHeight && blockHeight != 0) {
                // 结算稳定费
                uint256 fee = getFee(parassetAssets, blockHeight, pLedger.rate);
                // 转入p资产
                ERC20(pToken).safeTransferFrom(address(msg.sender), address(insurancePool),
    fee);
                // 消除负账户
                insurancePool.eliminate(pToken, pTokenToUnderlying[pToken]);
                emit FeeValue(pToken, fee);
          }
          ......
```

```
        }
```

The same issue exists in function `decrease()`、`increaseCoinage()` and `reducedCoinage()`.

Alleviation:

The development team heeded our advice and resolved this issue in commit

71bf95eb56af3bd21acfa685401e21bfccbe4eed

MP-03: Redundant Codes

Type	Severity	Location
Dead Code	Informational	MortgagePool.sol, L387,L427,L463,L499

Description:

Debt positions are already created in the function of `coin()`, and do not need to be checked again.

```
    function supplement(address mortgageToken,
                        address pToken,
                        uint256 amount) public payable whenActive {
          ......
            if (pLedger.created == false) {
                ledgerArray[pToken][mortgageToken].push(address(msg.sender));
                pLedger.created = true;
            }
        }
    function decrease(address mortgageToken,
                      address pToken,
                      uint256 amount) public payable whenActive {
          ......
            if (pLedger.created == false) {
                ledgerArray[pToken][mortgageToken].push(address(msg.sender));
                pLedger.created = true;
            }
        }
    function increaseCoinage(address mortgageToken,
                             address pToken,
                             uint256 amount) public payable whenActive {
           ......
            if (pLedger.created == false) {
                ledgerArray[pToken][mortgageToken].push(address(msg.sender));
                pLedger.created = true;
            }
        }
    function reducedCoinage(address mortgageToken,
                            address pToken,
                            uint256 amount) public payable whenActive {
           ......
            if (pLedger.created == false) {
                ledgerArray[pToken][mortgageToken].push(address(msg.sender));
                pLedger.created = true;
            }
        }
```

Recommendation:

Remove the following codes, and add the condition of `pLedger.created == true` in the beginning. An example revision is shown below:

```
    function supplement(address mortgageToken,
                        address pToken,
                        uint256 amount) public payable whenActive {
        require(pLedger.created,"Log:MortgagePool:!created");
      ......
        // Remove the following codes
        //if (pLedger.created == false) {
            //ledgerArray[pToken][mortgageToken].push(address(msg.sender));
            //pLedger.created = true;
        //}
    }
    function decrease(address mortgageToken,
                      address pToken,
                      uint256 amount) public payable whenActive {
      require(pLedger.created,"Log:MortgagePool:!created");
        ......
        // Remove the following codes
        //if (pLedger.created == false) {
            //ledgerArray[pToken][mortgageToken].push(address(msg.sender));
            //pLedger.created = true;
        //}
    }
    function increaseCoinage(address mortgageToken,
                             address pToken,
                             uint256 amount) public payable whenActive {
        ......
        // Remove the following codes
        //if (pLedger.created == false) {
            //ledgerArray[pToken][mortgageToken].push(address(msg.sender));
            //pLedger.created = true;
        //}
    }
    function reducedCoinage(address mortgageToken,
                            address pToken,
                            uint256 amount) public payable whenActive {
       require(pLedger.created,"Log:MortgagePool:!created");
        ......
        // Remove the following codes
        //if (pLedger.created == false) {
            //ledgerArray[pToken][mortgageToken].push(address(msg.sender));
            //pLedger.created = true;
        //}
    }
```

Alleviation:

The development team heeded our advice and resolved this issue in commit

71bf95eb56af3bd21acfa685401e21bfccbe4eed

MP-04: Fee Loss

Type	Severity	Location
Logical Issue	🔵Minor	MortgagePool.sol L565

Description:

There is no fee charged for liquidating:

```
    function liquidation(address mortgageToken,
                         address pToken,
                         address account) public payable whenActive {
          ......
          if (parassetAssets > 0 && block.number > blockHeight && blockHeight != 0) {
                fee = getFee(parassetAssets, blockHeight, pLedger.rate);
          }
           ......
          // 转入P资产
          ERC20(pToken).safeTransferFrom(address(msg.sender), address(insurancePool),
    pTokenAmount);
          ......
        }
```

Recommendation:

An example revision is shown below:

```
    function liquidation(address mortgageToken,
                         address pToken,
                         address account) public payable whenActive {
          ......
          if (parassetAssets > 0 && block.number > blockHeight && blockHeight != 0) {
                fee = getFee(parassetAssets, blockHeight, pLedger.rate);
          }
           ......
          // 转入P资产
          ERC20(pToken).safeTransferFrom(address(msg.sender), address(insurancePool),
    pTokenAmount.add(fee));
          ......
        }
```

Alleviation:

The team heeded our advice and added related check. Code change was applied in commit [71bf95eb56af3bd21acfa685401e21bfccbe4eed]

MP-05: Ever-growing LedgerArray

Type	Severity	Location
Logical Issue	⚪Discussion	MortgagePool.sol L509, L550

Description:

The `ledgerArray` should pop up after someone redeeming of all mortgages or liquidating.

```
    function redemptionAll(address mortgageToken, address pToken) public onleRedemptionAll {
           .......
           // Remove the following codes, and add pop-up operation.
            //if (pLedger.created == false) {
                //ledgerArray[pToken][mortgageToken].push(address(msg.sender));
                //pLedger.created = true;
            //}
        }
```

Alleviation:

The team heeded our advice and added related check. Code change was applied in commit 71bf95eb56af3bd21acfa685401e21bfccbe4eed

MP-06: Transfer Failure

Type	Severity	Location
Language Specific	Minor	MortgagePool.sol L425, L537,L584 IsurancePool.sol L227, L337, L339

Description:

Due to the 2300 gas limitation of the `transfer()` function, these following functions may fail when called through a contract:

```
// these functions located on MortgagePool.sol
function decrease(address mortgageToken,
                  address pToken,
                  uint256 amount) public payable whenActive {
      ......
      if (mortgageToken != address(0x0)) {
        ERC20(mortgageToken).safeTransfer(address(msg.sender), amount);
      } else {
        payEth(address(msg.sender), amount);
      }
       ......
    }
......
function redemptionAll(address mortgageToken, address pToken) public onleRedemptionAll {
     ......
        if (mortgageToken != address(0x0)) {
            ERC20(mortgageToken).safeTransfer(address(msg.sender), mortgageAssetsAmount);
        } else {
            payEth(address(msg.sender), mortgageAssetsAmount);
        }
     ......
    }
......
function liquidation(address mortgageToken,
                     address pToken,
                     address account) public payable whenActive {
      ......
      if (mortgageToken != address(0x0)) {
        ERC20(mortgageToken).safeTransfer(address(msg.sender), mortgageAssets);
      } else {
        payEth(address(msg.sender), mortgageAssets);
      }
    }
```

Recommendation:

Using the TransferHelper library. An example revision is shown below:

```
    import './lib/TransferHelper.sol';

    ......

    // these functions located on MortgagePool.sol
    function decrease(address mortgageToken,
                          address pToken,
                          uint256 amount) public payable whenActive {
          ......
          if (mortgageToken != address(0x0)) {
            ERC20(mortgageToken).safeTransfer(address(msg.sender), amount);
          } else {
                safeTransferETH(address(msg.sender), amount);
          }
           ......
        }
    ......
    function redemptionAll(address mortgageToken, address pToken) public onleRedemptionAll {
         ......
            if (mortgageToken != address(0x0)) {
                ERC20(mortgageToken).safeTransfer(address(msg.sender), mortgageAssetsAmount);
            } else {
                safeTransferETH(address(msg.sender), mortgageAssetsAmount);
            }
         ......
        }
    ......
    function liquidation(address mortgageToken,
                             address pToken,
                             address account) public payable whenActive {
          ......
          if (mortgageToken != address(0x0)) {
            ERC20(mortgageToken).safeTransfer(address(msg.sender), mortgageAssets);
          } else {
                safeTransferETH(address(msg.sender), mortgageAssets);
          }
        }
```

The same problem exists in InsurancePool smart contract.

Alleviation:

The development team heeded our advice and resolved this issue in commit

71bf95eb56af3bd21acfa685401e21bfccbe4eed

PC-01: Return Zero

Type	Severity	Location
Volatile Code	Informational	PriceController.sol L62

Description:

The price oracle from other contracts, `TokenPrice` and `pTokenPrice` , cannot be zero, otherwise they may introduce unpredictable vulnerabilities. One recommended approach is to add checks on the return value in `getPriceForPToken()` to ensure non-zero prices.

Alleviation:

The development team heeded our advice and resolved this issue in commit

71bf95eb56af3bd21acfa685401e21bfccbe4eed

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

 : Issue resolved

 : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

 : Issue partially resolved. Not all instances of an issue was resolved.