Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.11.09, the SlowMist security team received the Sperax team's security audit application for USDs, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability

- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Audit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

USDs is an algorithmic, highly scalable, trustless, decentralized stablecoin protocol that operates fully on chain. Most of the existing algorithmic stablecoins are highly trustless and scalable, but hard to bootstrap and tend to experience periods of high volatility. USDs plans to resolve these issues by borrowing insights from crypto collateralized cryptocurrencies like DAI, and dual token algorithmic stablecoins like Terra.

Audit Version:

https://github.com/Sperax/USDs/contracts

commit:d36fb1533554aafa431c0439aab139ded958ab04

Fixed Version:

https://github.com/Sperax/USDs/tree/slowmist-audit3

commit: 8dc36029823e075e4749e0d1be0916d98b149059

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Slippage check issue	Reordering Vulnerability	Medium	Confirmed
N2	Missing event record	Others	Suggestion	Fixed
N3	Price calculation issue	Design Logic Audit	Medium	Ignored
N4	Token release issue	Design Logic Audit	Suggestion	Confirmed
N5	Risk of excessive authority	Authority Control Vulnerability	Medium	Confirmed
N6	Logic defect	Design Logic Audit	Medium	Fixed
N7	Same algorithm issue	Design Logic Audit	Low	Fixed
N8	Overflow issue	Integer Overflow and Underflow Vulnerability	High	Fixed
N9	Redundant function	Others	Suggestion	Confirmed
N10	Sandwich attack risk	Design Logic Audit	Low	Fixed
N11	Contract docking defects	Design Logic Audit	Medium	Fixed
N12	Redundant initialize function	Others	Low	Fixed
N13	Balance check issue	Design Logic Audit	Low	Fixed
N14	`remove_liquidity_one_coin` issue	Design Logic Audit	Low	Fixed

NO	Title	Category	Level	Status
N15	Flashloan attack risk	Design Logic Audit	Critical	Fixed

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

BuybackMultihop			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
swap	External	Can Modify State	onlyVault

BuybackSingle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
swap	External	Can Modify State	onlyVault

InitializableAbstractStrategy

InitializableAbstractStrategy			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
_initialize	Internal	Can Modify State	-
collectRewardToken	External	Can Modify State	onlyVault nonReentrant
setRewardTokenAddress	External	Can Modify State	onlyOwner
setRewardLiquidationThreshold	External	Can Modify State	onlyOwner
setInterestLiquidationThreshold	External	Can Modify State	onlyOwner
setPTokenAddress	External	Can Modify State	onlyOwner
removePToken	External	Can Modify State	onlyOwner
_setPTokenAddress	Internal	Can Modify State	-
transferToken	Public	Can Modify State	onlyOwner
_abstractSetPToken	Internal	Can Modify State	-
safeApproveAllTokens	External	Can Modify State	-
deposit	External	Can Modify State	-
depositAll	External	Can Modify State	-
withdraw	External	Can Modify State	-
withdrawToVault	External	Can Modify State	-
withdrawInterest	External	Can Modify State	-
withdrawAll	External	Can Modify State	-
checkBalance	External	-	-

InitializableAbstractStrategy			
checkInterestEarned	External	-	-
supportsAsset	External	-	-

ThreePoolStrategy			
Function Name	Visibility	Mutability	Modifiers
<Receive Ether>	External	Payable	-
<Fallback>	External	Payable	-
initialize	External	Can Modify State	initializer
collectRewardToken	External	Can Modify State	onlyVault nonReentrant
deposit	External	Can Modify State	onlyVault nonReentrant
depositAll	External	Can Modify State	onlyVaultOrOwner nonReentrant
withdraw	External	Can Modify State	onlyVault nonReentrant
withdrawInterest	External	Can Modify State	onlyVault nonReentrant
withdrawToVault	External	Can Modify State	onlyOwner nonReentrant
withdrawAll	External	Can Modify State	onlyVaultOrOwner nonReentrant
checkBalance	Public	-	-
checkInterestEarned	Public	-	-
supportsAsset	External	-	-
safeApproveAllTokens	External	Can Modify State	-
_getTotalPTokens	Internal	-	-

ThreePoolStrategy			
_abstractSetPToken	Internal	Can Modify State	-
_getPoolCoinIndex	Internal	-	-

Oracle			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
updateUSDsAddress	External	Can Modify State	onlyOwner
updateVaultAddress	External	Can Modify State	onlyOwner
updateOraclePoolsAddress	External	Can Modify State	onlyOwner
changePeriod	External	Can Modify State	onlyOwner
updateCollateralInfo	External	Can Modify State	onlyOwner
updateInOutRatio	External	Can Modify State	-
getCollateralPrice	External	-	-
getETHprice	External	-	-
getSPAprice	External	-	-
getUSDsPrice	External	-	-
getUSDsPrice_average	External	-	-
getCollateralPrice_prec	External	-	-
getETHprice_prec	External	-	-
getSPAprice_prec	External	-	-

Oracle			
getUSDsPrice_prec	External	-	-
_getCollateralPrice	Internal	-	-
_getETHprice	Internal	-	-
_getCollateralPrice_prec	Internal	-	-

SperaxTokenL2			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20
setMintable	Public	Can Modify State	onlyOwner
revokeAllMintable	Public	Can Modify State	onlyOwner
mintForUSDs	External	Can Modify State	whenMintNotPaused onlyMintableGroup
burn	Public	Can Modify State	-
burnFrom	Public	Can Modify State	-
timelockOf	Public	-	-
transferWithLock	Public	Can Modify State	onlyOwner
release	Public	Can Modify State	onlyOwner
pause	Public	Can Modify State	onlyOwner
unpause	Public	Can Modify State	onlyOwner
mintPause	Public	Can Modify State	onlyOwner
mintUnpause	Public	Can Modify State	onlyOwner

SperaxTokenL2			
batchTransfer	Public	Can Modify State	-
_beforeTokenTransfer	Internal	Can Modify State	-
changeArbToken	External	Can Modify State	onlyOwner
bridgeMint	External	Can Modify State	onlyGateway
bridgeBurn	External	Can Modify State	onlyGateway

USDsL1			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
balanceOf	Public	-	-
transferFrom	Public	Can Modify State	-
isArbitrumEnabled	External	-	-
changeArbToken	External	Can Modify State	onlyOwner
registerTokenOnL2	Public	Can Modify State	-

USDsL2			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
changeVault	External	Can Modify State	onlyOwner
totalSupply	Public	-	-
balanceOf	Public	-	-

USDsL2			
creditsBalanceOf	Public	-	-
transfer	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-
_executeTransfer	Internal	Can Modify State	-
allowance	Public	-	-
approve	Public	Can Modify State	-
increaseAllowance	Public	Can Modify State	-
decreaseAllowance	Public	Can Modify State	-
mint	External	Can Modify State	onlyVault
_mint	Internal	Can Modify State	nonReentrant
burn	External	Can Modify State	onlyVault
_burn	Internal	Can Modify State	nonReentrant
_creditsPerToken	Internal	-	-
_isNonRebasingAccount	Internal	Can Modify State	-
_ensureRebasingMigration	Internal	Can Modify State	-
rebaseOptIn	Public	Can Modify State	onlyOwner nonReentrant
rebaseOptOut	Public	Can Modify State	onlyOwner nonReentrant
changeSupply	External	Can Modify State	onlyVault nonReentrant
changeArbToken	External	Can Modify State	onlyOwner
bridgeMint	External	Can Modify State	onlyGateway

USDsL2			
bridgeBurn	External	Can Modify State	onlyGateway

BancorFormula			
Function Name	Visibility	Mutability	Modifiers
initMaxExpArray	Private	Can Modify State	-
init	Public	Can Modify State	-
power	Public	-	-
generalLog	Internal	-	-
floorLog2	Internal	-	-
findPositionInMaxExpArray	Internal	-	-
generalExp	Internal	-	-
optimalLog	Internal	-	-
optimalExp	Internal	-	-

VaultCore			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
updateUSDsAddress	External	Can Modify State	onlyOwner
updateOracleAddress	External	Can Modify State	onlyOwner
updateParameters	External	Can Modify State	onlyOwner

VaultCore			
updateMintBurnPermission	External	Can Modify State	onlyOwner
updateAllocationPermission	External	Can Modify State	onlyOwner
updateSwapInOutFeePermission	External	Can Modify State	onlyOwner
addCollateral	External	Can Modify State	onlyOwner
updateCollateralInfo	External	Can Modify State	onlyOwner
addStrategy	External	Can Modify State	onlyOwner
mintWithUSDs	Public	Can Modify State	whenMintRedeemAllowed nonReentrant
mintWithSPA	Public	Can Modify State	whenMintRedeemAllowed nonReentrant
mintWithColla	Public	Can Modify State	whenMintRedeemAllowed nonReentrant
_mint	Internal	Can Modify State	whenMintRedeemAllowed
redeem	Public	Can Modify State	whenMintRedeemAllowed nonReentrant
_redeem	Internal	Can Modify State	whenMintRedeemAllowed
rebase	External	Can Modify State	whenRebaseAllowed nonReentrant
_harvest	Internal	Can Modify State	-
_harvestReward	Internal	Can Modify State	-
_harvestInterest	Internal	Can Modify State	-

VaultCore			
allocate	External	Can Modify State	whenAllocationAllowed onlyOwner nonReentrant
collateralRatio	Public	-	-
totalValueLocked	Public	-	-
totalValueInVault	Public	-	-
_valueInVault	Internal	-	-
totalValueInStrategies	Public	-	-
_valueInStrategy	Internal	-	-

VaultCoreTools			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
chiTarget	Public	-	-
multiplier	Public	-	-
chiMint	Public	-	-
chiRedeem	Public	-	-
calculateSwapFeeOut	Public	-	-
redeemView	Public	-	-
calculateSwapFeeIn	Public	-	-
mintView	Public	-	-
collaDeptAmountCalculator	Public	-	-

VaultCoreTools			
SPAAmountCalculator	Public	-	-
USDsAmountCalculator	Public	-	-

4.3 Vulnerability Summary

[N1] [Medium] Slippage check issue

Category: Reordering Vulnerability

Content

In the BuybackMultihop and BuybackSingle contracts, Vault can perform buyback operations through the swap function. However, the amountOutMinimum parameter passed in during token exchange through UniswapV3 is 0. Since the buyback operation is performed during rebase, there is a risk of sandwich attack.

Code location:

contracts/buyback/BuybackSingle.sol

```
    function swap(uint256 amountIn) external onlyVault override returns (uint256
amountOut) {
        TransferHelper.safeTransferFrom(inputToken, msg.sender, address(this),
amountIn);
        TransferHelper.safeApprove(inputToken, address(swapRouter), amountIn);
        ISwapRouter.ExactInputSingleParams memory params =
            ISwapRouter.ExactInputSingleParams({
                tokenIn: inputToken,
                tokenOut: USDs,
                fee: poolFee,
                recipient: msg.sender,
                deadline: block.timestamp,
                amountIn: amountIn,
                amountOutMinimum: 0,
                sqrtPriceLimitX96: 0
            });
        amountOut = swapRouter.exactInputSingle(params);
```

```
        IERC20(USDs).safeTransfer(vaultAddr, amountOut);
    }
```

contracts/buyback/BuybackMultihop.sol

```
    function swap(uint256 amountIn) external onlyVault override returns (uint256
amountOut) {
        TransferHelper.safeTransferFrom(inputToken, msg.sender, address(this),
amountIn);
        TransferHelper.safeApprove(inputToken, address(swapRouter), amountIn);
        ISwapRouter.ExactInputParams memory params =
            ISwapRouter.ExactInputParams({
                path: abi.encodePacked(inputToken, poolFee1, intermediateToken,
poolFee2, USDs),
                recipient: msg.sender,
                deadline: block.timestamp,
                amountIn: amountIn,
                amountOutMinimum: 0
            });
        amountOut = swapRouter.exactInput(params);
        IERC20(USDs).safeTransfer(vaultAddr, amountOut);
    }
```

Solution

It is recommended to check slippage when performing swap operations.

Status

Confirmed

[N2] [Suggestion] Missing event record

Category: Others

Content

In the Oracle contract, the owner can modify the sensitive parameters of the oracle through updateUSDsAddress, updateVaultAddress and updateOraclePoolsAddress, but no event recording is performed.

In the SperaxTokenL2 contract, the owner can set the l2Gateway and l1Address parameters through the changeArbToken function, but no event recording is performed.

Code location: contracts/oracle/Oracle.sol

```
    function updateUSDsAddress(address _USDsAddr) external onlyOwner {
        USDsAddr = _USDsAddr;
    }

    function updateVaultAddress(address _VaultAddr) external onlyOwner {
        VaultAddr = _VaultAddr;
    }

    function updateOraclePoolsAddress(address _SPAoracleBaseTokenAddr, address
_USDsOracleBaseTokenAddr, address _USDsOraclePool, address _SPAoraclePool) external
onlyOwner {
        SPAoracleBaseTokenAddr = _SPAoracleBaseTokenAddr;
        USDsOracleBaseTokenAddr = _USDsOracleBaseTokenAddr;
        USDsOraclePool = _USDsOraclePool;
        SPAoraclePool = _SPAoraclePool;
    }
```

contracts/token/SperaxTokenL2.sol

```
    function changeArbToken(address newL2Gateway, address newL1Address) external
onlyOwner {
        l2Gateway = newL2Gateway;
        l1Address = newL1Address;
    }
```

Solution

It is recommended to record incidents when modifying sensitive parameters of the contract for follow-up self-examination or community review.

Status

Fixed

[N3] [Medium] Price calculation issue

Category: Design Logic Audit

Content

In the oracle contract, users can get the price of SPA tokens through the getSPAprice function. However, the ETH price is used in the price calculation.

Code location: contracts/oracle/Oracle.sol

```
    function getSPAprice() external view override returns (uint) {
        uint32 longestSec =
OracleLibrary.getOldestObservationSecondsAgo(SPAoraclePool);
        uint32 period = movingAvgShortPeriod < longestSec ? movingAvgShortPeriod :
longestSec;
        int24 timeWeightedAverageTick = OracleLibrary.consult(SPAoraclePool, period);
        uint quoteAmount = OracleLibrary.getQuoteAtTick(timeWeightedAverageTick,
uint128(SPAprice_prec), SPAaddr, SPAoracleBaseTokenAddr);
        uint SPAprice = _getETHprice().mul(quoteAmount).div(ETHprice_prec);
        return SPAprice;
    }
```

Solution

It is recommended to use SPA accuracy for calculation

Status

Ignored; After communicating with the project team, the project team stated that the accuracy of the getSPAprice function using ETH is the expected design.

[N4] [Suggestion] Token release issue

Category: Design Logic Audit

Content

In the SperaxTokenL2 contract, when the user transfers tokens, the user's timelock status is checked through the _beforeTokenTransfer function. When the user satisfies `timelock.releaseTime != 0 &&`

`balanceOf(from).sub(amount) <timelock.amount`, if the user's releaseTime is less than or equal to the current time, the contract will update the user's timelock.amount state. However, since the user has met the condition of `block.timestamp >= timelock.releaseTime`, the user's tokens should have been unlocked, so the user's timelock.releaseTime and timelock.amount status should be set to 0 directly.

Code location: contracts/token/SperaxTokenL2.sol

```
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal
override {
        require(!paused(), "SperaxToken: token transfer while paused");

        TimeLock storage timelock = _timelock[from];
        if(timelock.releaseTime != 0 && balanceOf(from).sub(amount) <
timelock.amount) {
            require(block.timestamp >= timelock.releaseTime, "SperaxToken: current
time is before from account release time");

            timelock.amount = balanceOf(from).sub(amount);
            if(timelock.amount == 0) {
                timelock.releaseTime = 0;
            }
        }

        super._beforeTokenTransfer(from, to, amount);
    }
```

Solution

When the user satisfies the condition of `block.timestamp >= timelock.releaseTime`, it is recommended to directly set the user's timelock.releaseTime and timelock.amount status to 0.

Status

Confirmed

[N5] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

In the SperaxTokenL2 contract, the owner can add any user to the mintableAccounts list through the setMintable function, and set the mintableGroup status for this user. The user whose mintableGroup is true can mint tokens arbitrarily through the mintForUSDs function, which will lead to the risk of excessive owner authority.

In the SperaxTokenL2 contract, the owner can arbitrarily modify the l2Gateway address through the changeArbToken function. And l2Gateway can mint tokens arbitrarily through the bridgeMint function, or burn tokens of any user through the bridgeBurn function, which will lead to the risk of excessive owner authority.

Code location: contracts/token/SperaxTokenL2.sol

```
    function mintForUSDs(address account, uint256 amount) whenMintNotPaused
onlyMintableGroup external {
        _mint(account, amount);
    }

    function changeArbToken(address newL2Gateway, address newL1Address) external
onlyOwner {
        l2Gateway = newL2Gateway;
        l1Address = newL1Address;
    }
    modifier onlyGateway() {
        require(msg.sender == l2Gateway, "ONLY_l2GATEWAY");
        _;
    }

    function bridgeMint(address account, uint256 amount) external override
onlyGateway {
        _mint(account, amount);
    }

    function bridgeBurn(address account, uint256 amount) external override
onlyGateway {
        _burn(account, amount);
    }
```

Solution

It is recommended to transfer the ownership of the owner to community governance to avoid the risk of excessive authority.

Status

Confirmed; After communicating with the project team, the project team stated that the ownership of the owner will be controlled by the multisig contract in the early stage of the project and will be transferred to the DAO in the later stage of the project.

[N6] [Medium] Logic defect

Category: Design Logic Audit

Content

The redeemView function in the VaultCoreTools contract is used to calculate the corresponding redeem the value. When the incoming _collaAddr parameter is 0, it means that the collateral is a native token, but the price of the token in its if-esle logic is opposite to the judgment condition.

Code location: contracts/vault/VaultCoreTools.sol

```
        function redeemView(
                address _collaAddr,
                uint USDsAmt,
                address _VaultCoreContract,
                address _oracleAddr
        ) public view returns (uint SPAMintAmt, uint collaUnlockAmt, uint
USDsBurntAmt, uint swapFeeAmount) {

                ...

                if (_collaAddr == address(0)) {
                        collaUnlockAmt = multiplier(collaUnlockAmt,
IOracle(_oracleAddr).getCollateralPrice_prec(_collaAddr),
IOracle(_oracleAddr).getCollateralPrice(_collaAddr));
                        collaUnlockAmt = collaUnlockAmt.div(10**
(uint(18).sub(uint(ERC20Upgradeable(_collaAddr).decimals()))));
                } else {
```

```
                        collaUnlockAmt = multiplier(collaUnlockAmt,
IOracle(_oracleAddr).getETHprice_prec(), IOracle(_oracleAddr).getETHprice());
                }

            ....

        }
```

Solution

It is recommended to replace the if-else logic implementation.

Status

Fixed

[N7] [Low] Same algorithm issue

Category: Design Logic Audit

Content

In the VaultCoreTools contract, the collaDeptAmountCalculator function and the SPAAmountCalculator function are respectively used to calculate the amount of collateral that needs to be deposited and the amount of SPA tokens that need to be burned. It performs different calculations through the passed-in valueType parameter and if-else logic. But in actual implementation, the algorithm of if-else logic is the same. This will make the judgment of valueType meaningless.

Code location: contracts/vault/VaultCoreTools.sol

```
        function collaDeptAmountCalculator(
                uint valueType, uint USDsAmt, address _VaultCoreContract, address
collaAddr, uint swapFee
        ) public view returns (uint256 collaDeptAmt) {
                IVaultCore _vaultContract = IVaultCore(_VaultCoreContract);
                uint collaAddrDecimal = uint(ERC20Upgradeable(collaAddr).decimals());
                if (valueType == 1) {
                        collaDeptAmt =
USDsAmt.mul(chiMint(_VaultCoreContract)).mul(IOracle(_vaultContract.oracleAddr()).get
CollateralPrice_prec(collaAddr)).div(uint(_vaultContract.chi_prec()).mul(IOracle(_vau
```

```
ltContract.oracleAddr()).getCollateralPrice(collaAddr))).div(10**
(uint(18).sub(collaAddrDecimal)));
                        if (swapFee > 0) {
                                collaDeptAmt =
collaDeptAmt.add(collaDeptAmt.mul(swapFee).div(uint(_vaultContract.swapFee_prec())));
                        }
                } else if (valueType == 0) {
                        collaDeptAmt =
USDsAmt.mul(chiMint(_VaultCoreContract)).mul(IOracle(_vaultContract.oracleAddr()).get
CollateralPrice_prec(collaAddr)).div(uint(_vaultContract.chi_prec()).mul(IOracle(_vau
ltContract.oracleAddr()).getCollateralPrice(collaAddr))).div(10**
(uint(18).sub(collaAddrDecimal)));
                        if (swapFee > 0) {
                                collaDeptAmt =
collaDeptAmt.add(collaDeptAmt.mul(swapFee).div(uint(_vaultContract.swapFee_prec())));
                        }
                }
        }

        function SPAAmountCalculator(
                uint valueType, uint USDsAmt, address _VaultCoreContract, uint
swapFee
        ) public view returns (uint256 SPABurnAmt) {
                IVaultCore _vaultContract = IVaultCore(_VaultCoreContract);
                uint priceSPA = IOracle(_vaultContract.oracleAddr()).getSPAprice();
                uint precisionSPA =
IOracle(_vaultContract.oracleAddr()).getSPAprice_prec();
                if (valueType == 2 || valueType == 3) {
                        SPABurnAmt = USDsAmt.mul(uint(_vaultContract.chi_prec()) -
chiMint(_VaultCoreContract)).mul(precisionSPA).div(priceSPA.mul(uint(_vaultContract.c
hi_prec())));
                        if (swapFee > 0) {
                                SPABurnAmt =
SPABurnAmt.add(SPABurnAmt.mul(swapFee).div(uint(_vaultContract.swapFee_prec())));
                        }
                } else if (valueType == 0) {
                        SPABurnAmt = USDsAmt.mul(uint(_vaultContract.chi_prec()) -
chiMint(_VaultCoreContract)).mul(precisionSPA).div(priceSPA.mul(uint(_vaultContract.c
hi_prec())));
                        if (swapFee > 0) {
                                SPABurnAmt =
SPABurnAmt.add(SPABurnAmt.mul(swapFee).div(uint(_vaultContract.swapFee_prec())));
                        }
```

```
                }
        }
```

Solution

It is recommended to clarify the actual algorithm requirements.

Status

Fixed

[N8] [High] Overflow issue

Category: Integer Overflow and Underflow Vulnerability

Content

In the VaultCoreTools contract, the SPAAmountCalculator function and USDsAmountCalculator function respectively calculate the number of SPA tokens and USDs that need to be burned. When the valueType is 0, the SPAAmountCalculator function will calculate the number of SPA tokens through the algorithm, but because the calculation result of the chiMint function may be greater than `1e12`, the algorithm may have an overflow risk.

`USDsAmt.mul(uint(_vaultContract.chi_prec())-chiMint(_VaultCoreContract)).mul(precisionSPA).div(priceSPA.mul(uint(_vaultContract.chi_prec())));`

When the valueType is 1, the algorithm in the USDsAmountCalculator function also has this risk.

`USDsAmt.mul(uint(_vaultContract.chi_prec())).mul(priceSPA).div(precisionSPA.mul(uint(_vaultContract.chi_prec())-chiMint(_VaultCoreContract)));`

Code location: contracts/vault/VaultCoreTools.sol

```
        function SPAAmountCalculator(
                uint valueType, uint USDsAmt, address _VaultCoreContract, uint
swapFee
        ) public view returns (uint256 SPABurnAmt) {

...
```

```
                        SPABurnAmt = USDsAmt.mul(uint(_vaultContract.chi_prec()) -
chiMint(_VaultCoreContract)).mul(precisionSPA).div(priceSPA.mul(uint(_vaultContract.c
hi_prec())));
        }

        function USDsAmountCalculator(

...

            USDsAmt =
USDsAmt.mul(uint(_vaultContract.chi_prec())).mul(priceSPA).div(precisionSPA.mul(uint(
_vaultContract.chi_prec()) - chiMint(_VaultCoreContract)));
                }
        }
```

Solution

It is recommended to clearly design the algorithm to avoid such risks.

Status

Fixed

[N9] [Suggestion] Redundant function

Category: Others

Content

In the ThreePoolStrategy contract, the depositAll function and withdrawAll function do not have any specific implementation.

Code location: contracts/strategies/ThreePoolStrategy.sol

```
    function depositAll() external override onlyVaultOrOwner nonReentrant {}

    function withdrawAll() external override onlyVaultOrOwner nonReentrant {}
```

Solution

It is recommended to remove redundant functions.

Status

Confirmed

[N10] [Low] Sandwich attack risk

Category: Design Logic Audit

Content

In the ThreePoolStrategy contract, the Vault contract can raise funds in the strategy pool through the withdraw function, calculate the slippage through the calc_withdraw_one_coin function, and withdraw funds from the 3pool through the remove_liquidity_imbalance function, but there is still the risk of a sandwich attack.

The withdrawInterest function is the same as the withdrawToVault function

Code location: contracts/strategies/ThreePoolStrategy.sol

```
function withdraw(
    address _recipient,
    address _asset,
    uint256 _amount
) external override onlyVault nonReentrant {

    ...

    uint256 maxAmount = curvePool.calc_withdraw_one_coin(
        totalPTokens,
        int128(poolCoinIndex)
    );
    uint256 maxBurnedPTokens = totalPTokens.mul(_amount).div(maxAmount);

    // Not enough in this contract or in the Gauge, can't proceed
    require(totalPTokens > maxBurnedPTokens, "Insufficient 3CRV balance");
    // We have enough LP tokens, make sure they are all on this contract
    if (contractPTokens < maxBurnedPTokens) {
        // Not enough of pool token exists on this contract, some must be
        // staked in Gauge, unstake difference
        ICurveGauge(crvGaugeAddress).withdraw(
            maxBurnedPTokens.sub(contractPTokens)
        );
    }
```

```
        uint256[3] memory _amounts = [uint256(0), uint256(0), uint256(0)];
        _amounts[poolCoinIndex] = _amount;
        curvePool.remove_liquidity_imbalance(_amounts, maxBurnedPTokens);

        ...

    }
```

Solution

You can use an external oracle to feed the price and use this price to calculate slippage.

Status

Fixed; Fixed in the following repo:

https://github.com/Sperax/USDs/tree/slowmist-audit5

commit: fba2bb6ae0abd4aab616c61f565a368e5fe0b0cb

[N11] [Medium] Contract docking defects

Category: Design Logic Audit

Content

In the VaultCore contract, it will implement specific services by calling the supportsCollateral and rewardTokenBuybackAddress functions of the strategy contract, but these two functions are not implemented in the ThreePoolStrategy contract.

Code location: contarcts/vault/VaultCore.sol

```
        function _harvest() internal returns (uint USDsIncrement) {
                IStrategy strategy;
                collateralStruct memory collateral;
                for (uint y = 0; y < allCollaterals.length; y++) {
                        collateral = allCollaterals[y];
                        strategy = IStrategy(collateral.defaultStrategyAddr);
                        if (strategy.supportsCollateral(collateral.collateralAddr)
&& collateral.rebaseAllowed) {
                                uint USDsIncrement_viaReward =
```

```
_harvestReward(strategy);
                                uint USDsIncrement_viaInterest =
_harvestInterest(strategy, collateral.collateralAddr);
                                USDsIncrement =
USDsIncrement.add(USDsIncrement_viaReward).add(USDsIncrement_viaInterest);
                        }
                }
        }

        function _harvestReward(IStrategy strategy) internal returns (uint
USDsIncrement_viaReward) {
                address rewardTokenAddress = strategy.rewardTokenAddress();
        if (rewardTokenAddress != address(0)) {
            uint liquidationThreshold = strategy.rewardLiquidationThreshold();
                        uint rewardTokenAmount =
IERC20Upgradeable(rewardTokenAddress).balanceOf(address(this));
                        if (rewardTokenAmount > liquidationThreshold) {
                                strategy.collectRewardToken();
                                uint rewardAmt =
IERC20Upgradeable(rewardTokenAddress).balanceOf(address(this));

IERC20Upgradeable(rewardTokenAddress).safeTransfer(strategy.rewardTokenBuybackAddress
(), rewardAmt);
                                USDsIncrement_viaReward =
IBuyback(strategy.rewardTokenBuybackAddress()).swap(rewardTokenAddress, rewardAmt);
                        }
                }
        }
```

Solution

It is recommended to implement the necessary interfaces to meet business needs.

Status

Fixed

[N12] [Low] Redundant initialize function

Category: Others

Content

There is an initialize function in the InitializableAbstractStrategy contract, which initializes the parameters by calling

the `_initialize` function internally. But the InitializableAbstractStrategy contract is inherited by the ThreePoolStrategy contract, and the ThreePoolStrategy contract also has an initialize function, which also initializes parameters by internally calling the `_initialize` function. It is worth noting that the initialize function of ThreePoolStrategy does not use the override tag to rewrite the initialize function of the InitializableAbstractStrategy contract. This will result in two initialize functions in the same contract. If used incorrectly, it will cause repeated initialization problems.

Code location:

contracts/strategies/InitializableAbstractStrategy.sol

```
    function initialize(
        address _platformAddress,
        address _vaultAddress,
        address _rewardTokenAddress,
        address[] calldata _assets,
        address[] calldata _pTokens
    ) external initializer {
        OwnableUpgradeable.__Ownable_init();
        InitializableAbstractStrategy._initialize(
            _platformAddress,
            _vaultAddress,
            _rewardTokenAddress,
            _assets,
            _pTokens
        );
    }

    function _initialize(
        address _platformAddress,
        address _vaultAddress,
        address _rewardTokenAddress,
        address[] memory _assets,
        address[] memory _pTokens
    ) internal {
        platformAddress = _platformAddress;
        vaultAddress = _vaultAddress;
        rewardTokenAddress = _rewardTokenAddress;
        uint256 assetCount = _assets.length;
```

```
        require(assetCount == _pTokens.length, "Invalid input arrays");
        for (uint256 i = 0; i < assetCount; i++) {
            _setPTokenAddress(_assets[i], _pTokens[i]);
        }
    }
```

contracts/strategies/ThreePoolStrategy.sol

```
    function initialize(
        address _platformAddress, // 3Pool address
        address _vaultAddress,
        address _rewardTokenAddress, // CRV
        address[] calldata _assets,
        address[] calldata _pTokens,
        address _crvGaugeAddress
    ) external initializer {
        require(_assets.length == 3, "Must have exactly three assets");
        // Should be set prior to abstract initialize call otherwise
        // abstractSetPToken calls will fail
        crvGaugeAddress = _crvGaugeAddress;
        InitializableAbstractStrategy._initialize(
            _platformAddress,
            _vaultAddress,
            _rewardTokenAddress,
            _assets,
            _pTokens
        );
    }
```

Solution

It is recommended to remove the initialize function of the InitializableAbstractStrategy contract.

Status

Fixed

[N13] [Low] Balance check issue

Category: Design Logic Audit

Content

In the ThreePoolStrategy contract, the vault or owner can withdraw assets through the withdraw, withdrawInterest and withdrawToVault functions. It will first calculate how many Lp tokens (maxBurnedPTokens) need to be burned to extract the specified asset, and then check whether totalPTokens is greater than maxBurnedPTokens. If the amount that the user needs to advance is exactly equal to the totalPTokens value, the extraction will fail.

Code location: contracts/strategies/ThreePoolStrategy.sol

```
    function withdraw(
        address _recipient,
        address _asset,
        uint256 _amount
    ) external override onlyVault nonReentrant {
        ...

        uint256 maxAmount = curvePool.calc_withdraw_one_coin(
            totalPTokens,
            poolCoinIndex
        );
        uint256 maxBurnedPTokens = totalPTokens.mul(_amount).div(maxAmount);

        // Not enough in this contract or in the Gauge, can't proceed
        require(totalPTokens > maxBurnedPTokens, "Insufficient 3CRV balance");
        ...
    }
```

Solution

It is recommended to check whether totalPTokens is greater than or equal to maxBurnedPTokens

Status

Fixed; Fixed in the following repo:

https://github.com/Sperax/USDs/tree/slowmist-audit5

commit: fba2bb6ae0abd4aab616c61f565a368e5fe0b0cb

[N14] [Low] `remove_liquidity_one_coin` issue

Category: Design Logic Audit

Content

In the ThreePoolStrategy contract, the Vault or owner can withdraw assets through the withdraw, withdrawInterest and withdrawToVault functions. It will remove liquidity through the remove_liquidity_one_coin function, but it does not limit the `min_amount`, which will lead to the risk of a sandwich attack.

Code location: contracts/strategies/ThreePoolStrategy.sol

```
curvePool.remove_liquidity_one_coin(maxBurnedPTokens, poolCoinIndex, 0);
```

Solution

It is recommended to use the fair market rate of LP tokens denominated to calculate the minimum amount that needs to be withdrawn.

Status

Fixed; Fixed in the following repo:

https://github.com/Sperax/USDs/tree/slowmist-audit5

commit: fba2bb6ae0abd4aab616c61f565a368e5fe0b0cb

[N15] [Critical] Flashloan attack risk

Category: Design Logic Audit

Content

When the user performs redeem operations through the VaultCore contract, the VaultCore contract calls the withdraw function of the ThreePoolStrategy contract to withdraw the funds in the 3pool. When withdrawing funds, it will first calculate the maximum liquidity funds that can be withdrawn through the calc_withdraw_one_coin function, and then withdraw the funds required by the user through the remove_liquidity_one_coin function, but there is no slippage check.

If a malicious user performs a mint operation on the VaultCore contract in the balance state of 3pool, and then

flashloan causes a huge slippage in 3pool, and then the malicious user withdraws funds through the redeem function, the ThreePoolStrategy contract will calculate `maxBurnedPTokens` more than normal. This will cause part of ThreePoolStrategy's funds to be divided among other users of 3pool.

Code location: contracts/strategies/ThreePoolStrategy.sol

```
    function withdraw(
        address _recipient,
        address _asset,
        uint256 _amount
    ) external override onlyVault nonReentrant {
        require(_recipient != address(0), "Invalid recipient");
        require(_amount > 0, "Invalid amount");

        (uint256 contractPTokens, , uint256 totalPTokens) = _getTotalPTokens();

        uint256 poolCoinIndex = _getPoolCoinIndex(_asset);

        ICurvePool curvePool = ICurvePool(platformAddress);
        // Calculate how many platform tokens we need to withdraw the asset
        // amount in the worst case (i.e withdrawing all LP tokens)
        uint256 maxAmount = curvePool.calc_withdraw_one_coin(
            totalPTokens,
            poolCoinIndex
        );
        uint256 maxBurnedPTokens = totalPTokens.mul(_amount).div(maxAmount);

        // Not enough in this contract or in the Gauge, can't proceed
        require(totalPTokens > maxBurnedPTokens, "Insufficient 3CRV balance");
        // We have enough LP tokens, make sure they are all on this contract
        if (contractPTokens < maxBurnedPTokens) {
            // Not enough of pool token exists on this contract, some must be
            // staked in Gauge, unstake difference
            ICurveGauge(crvGaugeAddress).withdraw(
                maxBurnedPTokens.sub(contractPTokens)
            );
        }
        (contractPTokens, , ) = _getTotalPTokens();
        maxBurnedPTokens = maxBurnedPTokens < contractPTokens ? maxBurnedPTokens :
contractPTokens;
        uint256 balance_before = IERC20(_asset).balanceOf(address(this));
```

```
        curvePool.remove_liquidity_one_coin(maxBurnedPTokens, poolCoinIndex, 0);
        uint256 balance_after = IERC20(_asset).balanceOf(address(this));
        uint256 _amount_received = balance_after.sub(balance_before);
        if (_amount_received >= allocatedAmt[_asset]) {
            allocatedAmt[_asset] = 0;
        } else {
            allocatedAmt[_asset] = allocatedAmt[_asset].sub(_amount_received);
        }

        IERC20(_asset).safeTransfer(_recipient, _amount_received);
        emit Withdrawal(_asset, address(assetToPToken[_asset]), _amount_received);
    }
```

Solution

It is recommended to check slippage through get_virtual_price.

Status

Fixed; Fixed in the following repo:

https://github.com/Sperax/USDs/tree/slowmist-audit5

commit: fba2bb6ae0abd4aab616c61f565a368e5fe0b0cb

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002112090002	SlowMist Security Team	2021.11.09 - 2021.12.09	Medium Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 critical risk, 1 high risk, 5 medium risks, 5 low risks, 3 suggestion vulnerabilities. And 2 medium risk, 2 suggestion vulnerabilities were confirmed and being fixed; 1 medium risk vulnerabilities were ignored; All other findings were fixed. The code was not deployed to the mainnet.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist