Competitive Security Assessment

dappOS

June 5th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	dappOS
Platform & Language	Solidity
Codebase	• https://github.com/DappOSDao/contracts-v2 • audit commit - cb504c58d80d6aba643bf6dd67449c310c031f62 • final commit - 2b3b9e0359a0d607a9845b8b14c1d576587d41c8
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Code Vulnerability Review Summary

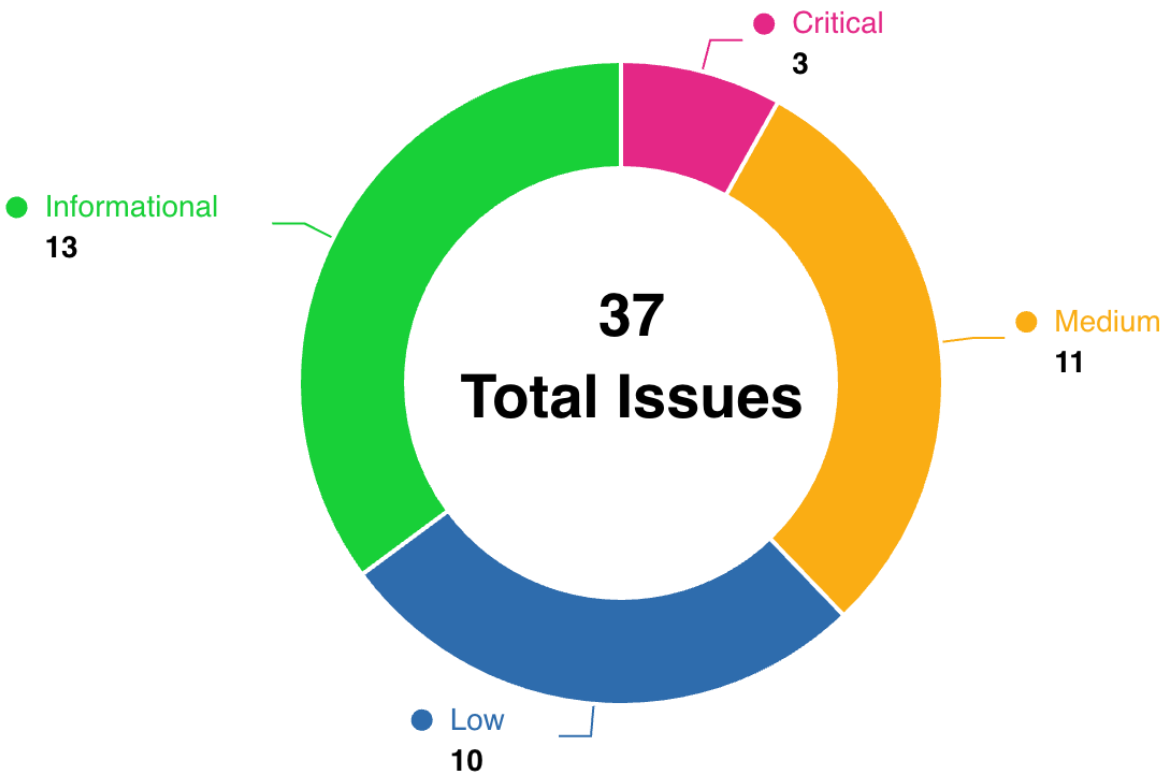
Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	3	0	0	2	0	1
Medium	11	0	3	4	0	4
Low	10	0	0	10	0	0
Informational	13	0	2	10	0	1

Audit Scope

File	Commit Hash
contracts/core/PayDB.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/libraries/BytesLib.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/vwmanager/VWManagerService.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/vwmanager/VWManager.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/peripherals/common/HybridPayService.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/governance/PayLock.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/peripherals/reward/RewardPool.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/interfaces/IPayDB.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/libraries/FullMath.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/libraries/HeaderLibrary.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/VirtualWallet.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/libraries/TransferHelper.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/vwmanager/VWManagersReader.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/libraries/VWCode.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/libraries/SignLibrary.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/peripherals/common/ERC20Service.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/libraries/SafeCast.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/interfaces/IVWManager.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/vwmanager/WalletDeployer.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/libraries/LowGasSafeMath.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/peripherals/reward/interfaces/IRewardPool.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/interfaces/IVWManagerStorage.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/interfaces/IVWManagersReader.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/interfaces/IVirtualWalletV2.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/vwmanager/storage/VWManagerStorage.sol	cb504c58d80d6aba643bf6dd67449c310c031f62

contracts/libraries/OrderId.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/interfaces/IVWResetter.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/interfaces/IVirtualWallet.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/interfaces/IWalletDeployer.sol	cb504c58d80d6aba643bf6dd67449c310c031f62
contracts/core/interfaces/IService.sol	cb504c58d80d6aba643bf6dd67449c310c031f62

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
DAP-1	Incorrect permission restrictions cause calls of functions `executeDstOrder()/executeDstOrderETH()` to fail	Logical	Critical	Fixed	Yaodao
DAP-2	User tokens probably stuck with an invalid node	Logical	Critical	Declined	danielt
DAP-3	malicious node can avoid being punished by frontrun	Logical	Critical	Fixed	jayphbee
DAP-4	DoS Attack of Unable to Become an Owner of a Wallet	Logical	Medium	Acknowledged	danielt

DAP-5	Front-runnning attack on `approve` function	Logical	Medium	Declined	jayphbee, danielt
DAP-6	Incorrect calculation of `gasFee`	Logical	Medium	Declined	Yaodao
DAP-7	Incorrect logic of volatileService	Logical	Medium	Declined	Yaodao
DAP-8	Lack of checking actual tokens received.	Logical	Medium	Fixed	danielt
DAP-9	No access control for `award` function	Privilege Related	Medium	Declined	jayphbee, Yaodao
DAP-10	Reentrancy risk in `PayLock` contract `claim` function	Reentrancy	Medium	Fixed	jayphbee, danielt, Yaodao, 0xzoobi
DAP-11	Use TransferHelper library to transfer ERC20.	Logical	Medium	Fixed	jayphbee, 0xzoobi
DAP-12	`_feeReceiver` is user controlled in the payFee function	Logical	Medium	Acknowledged	0xzoobi
DAP-13	`_wallet` and `realWallet` doesn't implement the `IVirtualWallet` interface	Logical	Medium	Acknowledged	jayphbee
DAP-14	for loop is early return in the `award` function	Logical	Medium	Fixed	jayphbee, danielt, Yaodao
DAP-15	Incorrect check in the function `submitWithdrawRequest()`	Logical	Low	Fixed	Yaodao
DAP-16	Incorrect check of deflationary token	Logical	Low	Fixed	Yaodao
DAP-17	Lack array parameters length equality check	Logical	Low	Fixed	jayphbee, Yaodao
DAP-18	Lack of checking result of the `ECDSA.recover` function	Signature Forgery or Replay	Low	Fixed	danielt
DAP-19	Lack of checks and updates to `result[wallet][codeToCancel]`	Logical	Low	Fixed	Yaodao
DAP-20	Lack of repeatability check	Logical	Low	Fixed	Yaodao

DAP-21	Potential Reentrancy risk in `VirtualWallet` contract `payFee` function	Reentrancy	Low	Fixed	hunya
DAP-22	Return value not checked.	Logical	Low	Fixed	jayphbee, Yaodao, 0xzoobi
DAP-23	Uncheck the result of the `VirtualWallet.execute()`	Code Style	Low	Fixed	Yaodao
DAP-24	VirtaulWallet should have the ability to receive ERC1155.	Logical	Low	Fixed	jayphbee, 0xzoobi
DAP-25	Cache array length outside of loop in `PayDB` contract	Gas Optimization	Informational	Fixed	hunya
DAP-26	Events are not indexed	Code Style	Informational	Fixed	0xzoobi
DAP-27	IService does not check return value on `execute`	Logical	Informational	Fixed	0xzoobi
DAP-28	Improve the error messages	Code Style	Informational	Fixed	Yaodao, 0xzoobi
DAP-29	Lack of zero address checking	Logical	Informational	Fixed	danielt
DAP-30	Missing emit event	Code Style	Informational	Fixed	danielt, 0xgm, 0xzoobi
DAP-31	Remove unnecessary receive or fallback function	Logical	Informational	Declined	jayphbee
DAP-32	Repeated functions in different contracts.	Gas Optimization	Informational	Acknowledged	danielt
DAP-33	Unused import	Code Style	Informational	Fixed	Yaodao
DAP-34	Unused return value	Logical	Informational	Fixed	danielt
DAP-35	`feeProportion` should be bounded.	Logical	Informational	Fixed	jayphbee
DAP-36	redundant use of `receive` and `fallback` in the same contract	Gas Optimization	Informational	Acknowledged	0xzoobi
DAP-37	use `external` identifier for functions instead of `public`	Gas Optimization	Informational	Fixed	Yaodao, 0xzoobi

DAP-1:Incorrect permission restrictions cause calls of functions `executeDstOrder()/executeDstOrderETH()` to fail

Category	Severity	Status	Contributor
Logical	Critical	Fixed	Yaodao

Code Reference

- code/contracts/core/VirtualWallet.sol#L37-L50
- code/contracts/core/PayDB.sol#L168-L237

```
37:     function execute(
38:         uint256 code,
39:         address service,
40:         bytes calldata data
41:     ) external override onlyVWManager returns (bool res) {
42:         (res, ) = service.delegatecall(
43:             abi.encodeWithSelector(
44:                 0x57c97782, //IService.execute.selector,
45:                 code,
46:                 data,
47:                 msg.sender
48:             )
49:         );
50:     }

168:     function _executeDstOrder(
169:         ExeOrderParam calldata eparam,
170:         address realWallet,
171:         bytes memory data,
172:         bytes memory serviceSignature
173:     ) internal {
174:         (uint256 dstChainId, , uint256 time) = OrderId.chainidsAndExpTime(
175:             eparam.payOrderId
176:         );
177:
178:         require(block.chainid == dstChainId && block.timestamp < time, "E8");
179:
180:         bytes32 workFlowHash = keccak256(
181:             abi.encode(
182:                 eparam.wallet,
183:                 eparam.code,
184:                 eparam.service,
185:                 keccak256(data)
186:             )
187:         );
188:         if (eparam.code != 0 && serviceSignature.length > 0) {
189:             if (eparam.manager == address(0)) {
190:                 IVirtualWallet(realWallet).execute(
191:                     eparam.code,
192:                     eparam.service,
193:                     data,
194:                     serviceSignature
```

```
195:                 );
196:             } else {
197:                 IVWManager.ExecuteParam memory exeParam = IVWManager
198:                     .ExecuteParam({
199:                         wallet: realWallet,
200:                         code: eparam.code,
201:                         service: eparam.service,
202:                         data: data,
203:                         proof: eparam.proof,
204:                         payToken: eparam.payToken,
205:                         gasTokenPrice: eparam.gasTokenPrice,
206:                         priorityFee: eparam.priorityFee,
207:                         gasLimit: eparam.gasLimit
208:                     });
209:
210:                 IVWManager(eparam.manager).execute(
211:                     exeParam,
212:                     serviceSignature,
213:                     eparam.isGateway,
214:                     eparam.feeReceiver
215:                 );
216:             }
217:         }
218:
219:         require(dstOrder[eparam.payOrderId] == 0, "E7");
220:         dstOrder[eparam.payOrderId] = keccak256(
221:             abi.encode(
222:                 eparam.amountOut,
223:                 eparam.tokenOut,
224:                 eparam.receiver,
225:                 workFlowHash
226:             )
227:         );
228:
229:         emit OrderExecuted(
230:             msg.sender,
231:             eparam.amountOut,
232:             eparam.tokenOut,
233:             eparam.receiver,
234:             eparam.payOrderId,
235:             workFlowHash
236:         );
237:     }
```

Description

Yaodao : The functions `executeDstOrder()/executeDstOrderETH()` deal the transfer of tokens and then call the function internal function `_executeDstOrder()`.In the function `PayDB._executeDstOrder()`, the function `IVirtualWallet(realWallet).execute()` will be called in the case `eparam.manager == address(0)`.

Consider below codes

```
    function _executeDstOrder(
        ExeOrderParam calldata eparam,
        address realWallet,
        bytes memory data,
        bytes memory serviceSignature
    ) internal {
        ...
        if (eparam.code != 0 && serviceSignature.length > 0) {
            if (eparam.manager == address(0)) {
                IVirtualWallet(realWallet).execute(
                    eparam.code,
                    eparam.service,
                    data,
                    serviceSignature
                );
            } else {
                ...
            }
        }
        ...
    }
```

However, in the contract `VirtualWallet`, the function `execute()` is declared with the modifier `onlyVWManager` which means the function `VirtualWallet.execute()` can be only called by the `VWManager` contract.

```
    function execute(
        uint256 code,
        address service,
        bytes calldata data
    ) external override onlyVWManager returns (bool res) {
        ...
    }
```

As a result, in the case `eparam.manager == address(0)`, the call of function `_executeDstOrder()` will always fail and the contract `PayDB` will not function properly.

Recommendation

Yaodao : Recommend considering the design of the corresponding logic and updating the logic.

Client Response

Fixed

DAP-2:User tokens probably stuck with an invalid node

Category	Severity	Status	Contributor
Logical	Critical	Declined	danielt

Code Reference

- code/contracts/governance/PayLock.sol#L74-L90

```
74:     function deposit(address token, uint amount, address node) external {
75:         TransferHelper.safeTransferFrom(token, msg.sender, address(this), amount);
76:         _deposit(token, amount, node);
77:     }
78:
79:     /// @notice deposit assets to get orders.
80:     /// Called by any service nodes
81:     function depositETH(address node) payable external {
82:         _deposit(address(0), msg.value, node);
83:     }
84:
85:     function _deposit(address token, uint amount, address node) internal {
86:         require(validTokens[token], "INVALID_TOKEN");
87:         TokenBalance storage bal = nodeTokenBalance[node][token];
88:         bal.numTotal = amount.add(bal.numTotal).toUint128();
89:         emit AssetsDeposited(node, token, amount);
90:     }
```

Description

danielt : In the `PayLock` contract, users deposit tokens into the contract and the corresponding amount is recorded to a specified node account that lacks checkings to ensure it is a valid node address. Note that the node intends to be a smart contract, the `submitWithdrawRequest` function tells us about it:

```
    function submitWithdrawRequest(address token, uint amount, address node) external returns (uint
requestID){
        if (msg.sender != node) {
            require(Ownable(node).owner() == msg.sender, "ONLY NODE/NODE_OWNER");
        }
        ...
    }
```

If a user deposits tokens to the `PayLock` contract with an EOA address as the node address, then the tokens of the user will be stuck in the contract since there is no validation on the `node` parameter in the `deposit` function and the `depos-`

`itETH` function.

Not limited to the above, the node accounts should be whitelisted by the project owner to prevent future unexpected behavior. It is because users can transfer tokens between users and the `PayLock` contract if the token is valid token.

Recommendation

danielt : Adding checkings on the `node` account in the `deposit` function and the `depositETH` function, including:

- ensuring the `node` account to be a valid contract with the `owner` function;
- checking if the node in a whitelist.

Client Response

Declined.EOA can withdraw money

DAP-3:malicious node can avoid being punished by frontrun

Category	Severity	Status	Contributor
Logical	Critical	Fixed	jayphbee

Code Reference

- code/contracts/governance/PayLock.sol#L113
- code/contracts/governance/PayLock.sol#L129

```
113:    function claim(address token, uint requestID) external {

129:    function punish(
```

Description

jayphbee : Node can deposit eth and erc20 to `PayLock` contract by calling `depositEth` and `deposit` and withdraw them by calling `submitWithrawRequest` then `claim`. There is a `token` field associated with the `WithdrawRequest`

```
struct WithdrawRequest {
        address token;
        ...
    }
```

The `claim` function doesn't validate the `token` parameter, however. This can lead to node accidently lose funds by calling `claim` with `token` parameter differentiate from the `token` filed in the `WithdrawRequest` corresponding to the `requestID`.

Here is the proof of concept test code.

```
pragma solidity ^0.8.0;

import "../../contracts/governance/PayLock.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";


import "forge-std/Test.sol";
import "forge-std/console2.sol";

contract MockERC20 is ERC20 {
    constructor(string memory name, string memory symbol) ERC20(name, symbol) {}

    function mint(address account, uint256 amount) public {
        _mint(account, amount);
    }
}

contract PayLockTest is Test {
    PayLock payLock;
    MockERC20 mockERC20;

    address node = address(1337);

    function setUp() public {
        payLock = new PayLock();
        mockERC20 = new MockERC20("MOCK", "MOCK");
        mockERC20.mint(node, 2000e18);

        payLock.configToken(address(mockERC20), true);
        payLock.configToken(address(0), true);
        vm.deal(node, 2000e18);
    }

    function testNodeStuckFunds() public {
        vm.startPrank(node);
        mockERC20.approve(address(payLock), type(uint256).max);

        payLock.deposit(address(mockERC20), 2000e18, node);
        payLock.depositETH{value: 2000e18}(node);

        // submit the withdraw request for __2000e18__ mockERC20
        payLock.submitWithdrawRequest(address(mockERC20), 2000e18, node); // requestId = 0
```

```
    // submit the withdraw request for __1000e18__ ether
        payLock.submitWithdrawRequest(address(0), 1000e18, node); // requestId = 1

        // NOTE: claim 1000 ether by requestId 0, instead of requestId 1
        payLock.claim(address(0), 0);
        // NOTE: claim 1000 mockERC20 by requestID 1
        payLock.claim(address(mockERC20), 1);

        console2.log("PayLock ether balance 1: ", address(payLock).balance);
        console2.log("PayLock erc20 balance 1: ", mockERC20.balanceOf(address(payLock)));

        // submit new withdraw request for the remainning 1000 ether
        payLock.submitWithdrawRequest(address(0), 1000e18, node); // requestId = 2
        payLock.claim(address(0), 2);

        console2.log("PayLock ether balance 2: ", address(payLock).balance);
        console2.log("PayLock erc20 balance 2: ", mockERC20.balanceOf(address(payLock)));

        // submit new withdraw request for the remainning 1000e18 mockERC20, this will revert
        vm.expectRevert("INSUFFICIENT_AMOUNT");
        payLock.submitWithdrawRequest(address(mockERC20), 1000e18, node);

        console2.log("PayLock ether balance 3: ", address(payLock).balance);
        console2.log("PayLock erc20 balance 3: ", mockERC20.balanceOf(address(payLock)));
    }
}
```

Run `forge test --mt testNodeStuckFunds -vvvv`

Logs: PayLock ether balance 1: 10000000000000000000 PayLock erc20 balance 1: 10000000000000000000 PayLock ether balance 2: 0 PayLock erc20 balance 2: 10000000000000000000 PayLock ether balance 3: 0 PayLock erc20 balance 3: 10000000000000000000

The test result shows that there are 1000e18 erc20 tokens stucked in the `PayLock` contract.

jayphbee : A malicious node will be by punished by calling `punish` function when t(t > n/2) super nodes reach an agreement, but the `punish` transaction can be frontrun by the node thus avoid being slashed some funds if the node can include its `submitWithdrawRequest` and `claim` transaction prior to the `punish` transaction.

Recommendation

jayphbee : Remove the `token` parameter and use `req.token` in the `claim` function.

```
function claim(uint requestID) external {
        WithdrawRequest storage req = withdrawRequests[requestID];
        require(req.status == 1 && req.submitTime + withdrawPendingTime <= block.timestamp, "CLAIM_P
ENDING");
        if (msg.sender != req.node) {
            require(Ownable(req.node).owner() == msg.sender, "ONLY NODE/NODE_OWNER");
        }
        TokenBalance storage bal = nodeTokenBalance[req.node][req.token]; // token => req.token
        uint numCanWithdraw = FullMath.min(bal.numOnWithdraw, req.amount);
        TransferHelper.safeTransfer2(req.token, req.node, numCanWithdraw); // token => req.token
        bal.numOnWithdraw = numCanWithdraw <= req.amount ? 0 : uint(bal.numOnWithdraw).sub(req.amoun
t).toUint128();
        req.status = 2;
        emit AssetsClaimed(requestID, req.node, numCanWithdraw, req.token); // token => req.token
    }
```

jayphbee : Add a time delay between `submitWithdrawRequest` and `claim`.

Client Response

Fixed

DAP-4:DoS Attack of Unable to Become an Owner of a Wallet

Category	Severity	Status	Contributor
Logical	Medium	Acknowledged	danielt

Code Reference

- code/contracts/core/vwmanager/VWManagerService.sol#L205

```
205:        require(ownerWallet[newOwner] == address(0) && newOwner != address(0), "E4");
```

Description

danielt : In the `VWManagerService` contract, each wallet has one owner, at the same time, each user only has one wallet.

As a result, the user Alice is able to assign a wallet to the new Owner Bob, which makes Bob unable to own a new wallet.

The impact is that malicious users can create new wallets to forbid others to own a wallet.

Recommendation

danielt : Recommend adding 2 steps logic, that is similar to `acceptOwnership` function in the `Ownable2Step.sol` contract to ensure that the new owner has the power to choose to become the owner of the new wallet or not.

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol

Client Response

Acknowledged.if B reset vw1 to A, A can transfer vw1 to others and then create vw of own

DAP-5:Front-runnning attack on `approve` function

Category	Severity	Status	Contributor
Logical	Medium	Declined	jayphbee, danielt

Code Reference

- code/contracts/peripherals/common/ERC20Service.sol#L27
- code/contracts/peripherals/common/HybridPayService.sol#L181

```
27:             IERC20(token).approve(spender, amount);

181:                 IERC20(cparams[i].tokenIn).approve(payDB, type(uint256).max);
```

Description

jayphbee : To approve some USDT to a certain address, it must ensure that the allowance to that address is 0, otheriwise approve will revert. As the code in ethereum mainnet shows:

```
function approve(address _spender, uint _value) public onlyPayloadSize(2 * 32) {

        // To change the approve amount you first have to reduce the addresses`
        //  allowance to zero by calling `approve(_spender, 0)` if it is not
        //  already 0 to mitigate the race condition described here:
        //  https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
        require(!((_value != 0) && (allowed[msg.sender][_spender] != 0)));

        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
    }
```

danielt : The front-running attack on the `approve` function is well known. The spender is able to detect the approving transaction to the spender from the approver and used up all the allowance the spender got before a new approving transaction is executed.

Recommendation

jayphbee : User `SafeERC20.sol` library of Openzeppelin to approve USDT to a certain address.

```
        if (p.action == 0) {
            // Approve
            (address token, address spender, uint256 amount) = abi.decode(
                data[32:],
                (address, address, uint256)
            );
            IERC20(token).safeApprove(spender, 0);
            IERC20(token).safeApprove(spender, amount);
        }
```

Apply the similar change to the `HybridPayService.sol#L181` as well.

danielt : Consider using the `increaseAllowance()` and `decreaseAllowance()` functions if the tokens are ERC20 tokens with the implementation that uses the OpenZeppelin library.

documents: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol#L177-L206

Client Response

Declined.PayDB can only spend msg.sender’s token and emit a event, it does not matter if user approve more allowence to paydb

DAP-6:Incorrect calculation of `gasFee`

Category	Severity	Status	Contributor
Logical	Medium	Declined	Yaodao

Code Reference

- code/contracts/core/vwmanager/VWManager.sol#L174-L215

```
174:     function execute(
175:         ExecuteParam calldata eParam,
176:         bytes calldata serviceSignature,
177:         bool isGateway,
178:         address feeReceiver
179:     ) external override returns (bool res) {
180:         uint256 preGas = gasleft();
181:         if (isGateway) {
182:             volatileService = eParam.service;
183:         }
184:         require(eParam.service != address(0), "E11");
185:         res = IVirtualWalletV2(eParam.wallet).execute(
186:             eParam.code,
187:             eParam.service,
188:             eParam.data
189:         );
190:         if (isGateway) {
191:             delete volatileService;
192:         }
193:         verifyProof(res, eParam, serviceSignature);
194:         uint256 gasUsed = preGas - gasleft();
195:         require(gasUsed <= eParam.gasLimit, "E13");
196:         uint256 gasFee = (eParam.gasTokenPrice * gasUsed) /
197:             1e8 +
198:             eParam.priorityFee;
199:
200:         if (protocolFeeOpened) {
201:             splitAndSendFee(
202:                 eParam.wallet,
203:                 eParam.payToken,
204:                 gasFee,
205:                 feeReceiver
206:             );
207:         } else {
208:             res = IVirtualWalletV2(eParam.wallet).payFee(
209:                 eParam.payToken,
210:                 gasFee,
211:                 feeReceiver
212:             );
213:         }
214:         require(res, "E10");
215:     }
```

Description

Yaodao : In contract `VWManager`, the function `execute` will calculate the gas cost using the following formula:

```
uint256 gasUsed = preGas - gasleft();
require(gasUsed <= eParam.gasLimit, "E13");
uint256 gasFee = (eParam.gasTokenPrice * gasUsed) /
            1e8 +
            eParam.priorityFee;
```

The `eParam.gasTokenPrice` is the price of gas, which is passed in externally. For EVM-compatible blockchains, the unit of the price of gas is generally `Gwei`, which is equal to 1e9 `wei`(Refer to the gas price in the doc:https://ethereum.org/en/developers/docs/gas/). The `gasUsed` is the gas you have used, which unit is `wei`. The `gasFee` should be calculated as the following calculation with converting the unit of `eParam.gasTokenPrice` to `wei`.

```
uint256 gasFee = (eParam.gasTokenPrice * gasUsed) /
            1e9 +
            eParam.priorityFee;
```

Recommendation

Yaodao : Consider below fix in the `execute` function

```
uint256 gasFee = (eParam.gasTokenPrice * gasUsed) /
            1e9 +
            eParam.priorityFee;
```

Client Response

Declined.The gasTokenPrice is converted externally based on the minimum precision of that token. The issue arises when the token is expensive but has insufficient precision. In such cases, the value represented by the minimum precision of that token in terms of USD will be significant. Therefore, we default to increasing the precision of each token by eight decimal places(1e8)

DAP-7:Incorrect logic of volatileService

Category	Severity	Status	Contributor
Logical	Medium	Declined	Yaodao

Code Reference

- code/contracts/core/vwmanager/storage/VWManagerStorage.sol#L24
- code/contracts/peripherals/reward/RewardPool.sol#L32
- code/contracts/core/VirtualWallet.sol#L37-L50
- code/contracts/core/vwmanager/VWManager.sol#L174-L215

```
24:    address public override volatileService;

32:            IVWManager(rewardConfig.gateway).volatileService() ==

37:    function execute(
38:        uint256 code,
39:        address service,
40:        bytes calldata data
41:    ) external override onlyVWManager returns (bool res) {
42:        (res, ) = service.delegatecall(
43:            abi.encodeWithSelector(
44:                0x57c97782, //IService.execute.selector,
45:                code,
46:                data,
47:                msg.sender
48:            )
49:        );
50:    }

174:    function execute(
175:        ExecuteParam calldata eParam,
176:        bytes calldata serviceSignature,
177:        bool isGateway,
178:        address feeReceiver
179:    ) external override returns (bool res) {
180:        uint256 preGas = gasleft();
181:        if (isGateway) {
182:            volatileService = eParam.service;
183:        }
184:        require(eParam.service != address(0), "E11");
185:        res = IVirtualWalletV2(eParam.wallet).execute(
186:            eParam.code,
187:            eParam.service,
188:            eParam.data
189:        );
190:        if (isGateway) {
191:            delete volatileService;
192:        }
193:        verifyProof(res, eParam, serviceSignature);
194:        uint256 gasUsed = preGas - gasleft();
195:        require(gasUsed <= eParam.gasLimit, "E13");
196:        uint256 gasFee = (eParam.gasTokenPrice * gasUsed) /
```

```
197:             1e8 +
198:             eParam.priorityFee;
199:
200:         if (protocolFeeOpened) {
201:             splitAndSendFee(
202:                 eParam.wallet,
203:                 eParam.payToken,
204:                 gasFee,
205:                 feeReceiver
206:             );
207:         } else {
208:             res = IVirtualWalletV2(eParam.wallet).payFee(
209:                 eParam.payToken,
210:                 gasFee,
211:                 feeReceiver
212:             );
213:         }
214:         require(res, "E10");
215:     }
```

Description

Yaodao : According to the following codes, the check will judge whether the `volatileService` of the `IVWManager(rewardConfig.gateway)` is the same as `rewardConfig.service`. Besides, the `rewardConfig.service` can't be `address(0)`.

```
    require(rewardConfig.service != address(0), "NO REWARD CONFIG");
    require(
        IVWManager(rewardConfig.gateway).volatileService() ==
            rewardConfig.service,
        "UNAUTHORIZED SERVICE"
    );
```

The `volatileService` in the contract `VWManager` is declared and the default value is `address(0)`. The following codes are all the logic related to `volatileService` in the contract `VWManager`. The `volatileService` will be updated to `eParam.service` when the `isGateway` is true. However, the value of `volatileService` will be deleted and back to `address(0)` in the later logic. Unless the call of `RewardPool.award()` is in the call of `IVirtualWalletV2(eParam.wallet).execute()`.

```
    if (isGateway) {
        volatileService = eParam.service;
    }
    require(eParam.service != address(0), "E11");
    res = IVirtualWalletV2(eParam.wallet).execute(
        eParam.code,
        eParam.service,
        eParam.data
    );
    if (isGateway) {
        delete volatileService;
    }
```

Besides, the implementation of the function `VirtualWallet.excute()` can't state that `RewardPool.award()` is called by it.

```
    function execute(
        uint256 code,
        address service,
        bytes calldata data
    ) external override onlyVWManager returns (bool res) {
        (res, ) = service.delegatecall(
            abi.encodeWithSelector(
                0x57c97782, //IService.execute.selector,
                code,
                data,
                msg.sender
            )
        );
    }
```

As a result, the call of function `RewardPool.award()` will always fail because the `IVWManager(rewardConfig.gateway).volatileService()` is always `address(0)` in the call and the `rewardConfig.service` can't be `address(0)`.

Recommendation

Yaodao : Recommend adding the function to set the address `volatileService` or updating the logic of check.

Client Response

Declined.reward contract is designed to only be called by vwmanager

DAP-8:Lack of checking actual tokens received.

Category	Severity	Status	Contributor
Logical	Medium	Fixed	danielt

Code Reference

- code/contracts/governance/PayLock.sol#L74-L77

```
74:     function deposit(address token, uint amount, address node) external {
75:         TransferHelper.safeTransferFrom(token, msg.sender, address(this), amount);
76:         _deposit(token, amount, node);
77:     }
```

Description

danielt : In the `PayLock` contract, users can deposit tokens to a `node` through the `deposit` function. However, in the `deposit` function, lacks checking the actual tokens the `PayLoack` contract received. Especially, if the token is a deflationary token, the actual tokens received will be less than the amount specified in the `safeTransferFrom` function and the number recorded in the `nodeTokenBalance`:

```
    function deposit(address token, uint amount, address node) external {
        TransferHelper.safeTransferFrom(token, msg.sender, address(this), amount);
        _deposit(token, amount, node);
    }
```

On the other side, users can invoke the `submitWithdrawRequest` function and `claim` function to withdraw tokens. If the node is the user and a deflationary token is in the `validTokens` mapping, then a user can exhaust the deflationary token of the `PayLock` contract with repeated deposits and withdrawals.

Recommendation

danielt : Record the balance of the `PayLock` contract for the token before and after the `safeTransferFrom` function, and calculate the increment as the actually received amount of tokens.

```
    function deposit(address token, uint amount, address node) external {
        uint before = IERC20(token).balanceOf(address(this));
        TransferHelper.safeTransferFrom(token, msg.sender, address(this), amount);
         uint after = IERC20(token).balanceOf(address(this));
        _deposit(token, after - before, node);
    }
```

Client Response

Fixed

DAP-9:No access control for `award` function

Category	Severity	Status	Contributor
Privilege Related	**Medium**	**Declined**	**jayphbee, Yaodao**

Code Reference

- code/contracts/peripherals/reward/RewardPool.sol#L25-L29
- code/contracts/peripherals/reward/RewardPool.sol#L25-L69

```
25:     function award(
26:         address[] calldata tokens,
27:         uint256[] calldata amounts,
28:         address to
29:     ) external override {
30:         require(rewardConfig.service != address(0), "NO REWARD CONFIG");
31:         require(
32:             IVWManager(rewardConfig.gateway).volatileService() ==
33:                 rewardConfig.service,
34:             "UNAUTHORIZED SERVICE"
35:         );
36:         require(tokens.length == amounts.length, "TOKE AMOUNT NO MATCH");
37:         for (uint256 i = 0; i < tokens.length; i++) {
38:             require(claimConfigs[tokens[i]].k > 0, "K NOT SET");
39:             if (rewardConfig.rewardToken != address(0)) {
40:                 TransferHelper.safeTransfer2(
41:                     rewardConfig.rewardToken,
42:                     to,
43:                     FullMath.min(
44:                         FullMath.mulDiv(
45:                             amounts[i],
46:                             claimConfigs[tokens[i]].k,
47:                             denonimator
48:                         ),
49:                         IERC20(rewardConfig.rewardToken).balanceOf(
50:                             address(this)
51:                         )
52:                     )
53:                 );
54:                 return;
55:             }
56:             TransferHelper.safeTransfer2(
57:                 rewardConfig.rewardToken,
58:                 to,
59:                 FullMath.min(
60:                     FullMath.mulDiv(
61:                         amounts[i],
62:                         claimConfigs[tokens[i]].k,
63:                         denonimator
64:                     ),
65:                     address(this).balance
66:                 )
```

```
67:             );
68:         }
69:     }

25:     function award(
26:         address[] calldata tokens,
27:         uint256[] calldata amounts,
28:         address to
29:     ) external override {
```

Description

jayphbee : There is no access control for the `award` function so that anyone can tranfer funds in `RewardPool` contract.

```
function award(
        address[] calldata tokens,
        uint256[] calldata amounts,
        address to
    ) external override {
    ....
    }
```

Yaodao : In the function `RewardPool.award()`, the `to` address is given via parameter. Refer to the issue `Incorrect logic of volatileService`, unless the design is that the function `RewardPool.award()` can only be called by `VirtualWallet.execute()` (called by the `VWManager.excute()`), the function `RewardPool.award()` lacks permission control and the attacker can get tokens from the contract.

Consider below codes

```
    function award(
        address[] calldata tokens,
        uint256[] calldata amounts,
        address to
    ) external override {
        require(rewardConfig.service != address(0), "NO REWARD CONFIG");
        require(
            IVWManager(rewardConfig.gateway).volatileService() ==
                rewardConfig.service,
            "UNAUTHORIZED SERVICE"
        );
        require(tokens.length == amounts.length, "TOKE AMOUNT NO MATCH");
        for (uint256 i = 0; i < tokens.length; i++) {
            require(claimConfigs[tokens[i]].k > 0, "K NOT SET");
            if (rewardConfig.rewardToken != address(0)) {
                TransferHelper.safeTransfer2(
                    rewardConfig.rewardToken,
                    to,
                    FullMath.min(
                        FullMath.mulDiv(
                            amounts[i],
                            claimConfigs[tokens[i]].k,
                            denonimator
                        ),
                        IERC20(rewardConfig.rewardToken).balanceOf(
                            address(this)
                        )
                    )
                );
                return;
            }
            TransferHelper.safeTransfer2(
                rewardConfig.rewardToken,
                to,
                FullMath.min(
                    FullMath.mulDiv(
                        amounts[i],
                        claimConfigs[tokens[i]].k,
                        denonimator
                    ),
                    address(this).balance
                )
            );
```

```
  }
      }
```

Recommendation

jayphbee : Add `onlyOwner` for the `award` function.
Yaodao : Recommend adding permission control or stating for this.

Client Response

Declined

DAP-10:Reentrancy risk in `PayLock` contract `claim` function

Category	Severity	Status	Contributor
Reentrancy	Medium	Fixed	jayphbee, danielt, Yaodao, 0xzoobi

Code Reference

- code/contracts/governance/PayLock.sol#L113-L125
- code/contracts/governance/PayLock.sol#L121

```
113:    function claim(address token, uint requestID) external {
114:        WithdrawRequest storage req = withdrawRequests[requestID];
115:        require(req.status == 1 && req.submitTime + withdrawPendingTime <= block.timestamp, "CLA
IM_PENDING");
116:        if (msg.sender != req.node) {
117:            require(Ownable(req.node).owner() == msg.sender, "ONLY NODE/NODE_OWNER");
118:        }
119:        TokenBalance storage bal = nodeTokenBalance[req.node][token];
120:        uint numCanWithdraw = FullMath.min(bal.numOnWithdraw, req.amount);
121:        TransferHelper.safeTransfer2(token, req.node, numCanWithdraw);
122:        bal.numOnWithdraw = numCanWithdraw <= req.amount ? 0 : uint(bal.numOnWithdraw).sub(req.a
mount).toUint128();
123:        req.status = 2;
124:        emit AssetsClaimed(requestID, req.node, numCanWithdraw, token);
125:    }

121:        TransferHelper.safeTransfer2(token, req.node, numCanWithdraw);
```

Description

jayphbee : `claim` is vunlerable to reentance attack because it doesn't follow the CEI(check-effect-interaction) pattern.

```
    function claim(address token, uint requestID) external {
        ...
        uint numCanWithdraw = FullMath.min(bal.numOnWithdraw, req.amount);
        TransferHelper.safeTransfer2(token, req.node, numCanWithdraw);
        bal.numOnWithdraw = numCanWithdraw <= req.amount ? 0 : uint(bal.numOnWithdraw).sub(req.amoun
t).toUint128();
        req.status = 2;
        emit AssetsClaimed(requestID, req.node, numCanWithdraw, token);
    }
```

If `token` is ETH `safeTransfer2` will make an external call to `req.node` and it can reenter to `claim` and claim more funds than expected.

danielt : In the `PayLock` contract, users can invoke the `claim` function to get the tokens. However, the implementation of the `claim` function neither matches the checks-effects-interactions-pattern nor uses a nonReentrant lock:

```
    function claim(address token, uint requestID) external {
        WithdrawRequest storage req = withdrawRequests[requestID];
        require(req.status == 1 && req.submitTime + withdrawPendingTime <= block.timestamp, "CLAIM_P
ENDING");
        if (msg.sender != req.node) {
            require(Ownable(req.node).owner() == msg.sender, "ONLY NODE/NODE_OWNER");
        }
        TokenBalance storage bal = nodeTokenBalance[req.node][token];
        uint numCanWithdraw = FullMath.min(bal.numOnWithdraw, req.amount);
        TransferHelper.safeTransfer2(token, req.node, numCanWithdraw);
        bal.numOnWithdraw = numCanWithdraw <= req.amount ? 0 : uint(bal.numOnWithdraw).sub(req.amoun
t).toUint128();
        req.status = 2;
        emit AssetsClaimed(requestID, req.node, numCanWithdraw, token);
    }
```

As a result, a user may reenter the `claim` function to drain tokens in the `PayLock` contract if the valid token is similar to the ERC 777 token with a `tokensReceived` function.

Yaodao : The following codes in the function `claim()` do not meet the Checks-Effects-Interactions pattern. The reentrancy attack can happen as the `req.status` is not updated before transfer when the caller is a contract. As a result, the caller can claim more tokens instead the expected amount.

Consider below codes

```
    function claim(address token, uint requestID) external {
        WithdrawRequest storage req = withdrawRequests[requestID];
        require(req.status == 1 && req.submitTime + withdrawPendingTime <= block.timestamp, "CLAIM_P
ENDING");
        if (msg.sender != req.node) {
            require(Ownable(req.node).owner() == msg.sender, "ONLY NODE/NODE_OWNER");
        }
        TokenBalance storage bal = nodeTokenBalance[req.node][token];
        uint numCanWithdraw = FullMath.min(bal.numOnWithdraw, req.amount);
        TransferHelper.safeTransfer2(token, req.node, numCanWithdraw);
        bal.numOnWithdraw = numCanWithdraw <= req.amount ? 0 : uint(bal.numOnWithdraw).sub(req.amoun
t).toUint128();
        req.status = 2;
        emit AssetsClaimed(requestID, req.node, numCanWithdraw, token);
    }
```

0xzoobi : Paylock's `claim` function is used to claim locked assets by any service nodes.

The Paylock's `claim` does not follow the `checks-effects-interaction` pattern. As a result, the variables `bal.numOnWithdraw` and `req.status` are updated post transfer, which results in Reentrancy.

The attacker can make an Reentrancy call via `fallback` function recursively and drain all the funds for the address of the token passed during `claim` function.

A malicious user can deposit any ERC20 token and then withdraw them all, basically draining the contract of all the funds.

Steps of Attack - An Example

1. Attacker deposits 1 WETH token via `deposit`.
2. Attacker calls `submitWithdrawRequest` for WETH to withdraw 1 WETH.
3. Attacker calls `claim` , the problem is the variables `bal.numOnWithdraw` and `req.status` are updated after calling `TransferHelper.safeTransfer2` , As a result, attacker can reenter the function and drain the contract of WETH token.

Note: The re-entrancy effects only the token address passed as part of the `claim` function but the attacker can make small deposits in all the tokens and `submitWithdrawRequest` for the tokens and drain the funds.

I am totally aware there is a `punish` function in place to punish the nodes, but the attacker can always `frontrun` the punish transactions and run away with the stolen funds.

Recommendation

jayphbee : Move `safeTransfer2` call at the end of `claim` function.

```
    function claim(address token, uint requestID) external {
        WithdrawRequest storage req = withdrawRequests[requestID];
        require(req.status == 1 && req.submitTime + withdrawPendingTime <= block.timestamp, "CLAIM_P
ENDING");
        if (msg.sender != req.node) {
            require(Ownable(req.node).owner() == msg.sender, "ONLY NODE/NODE_OWNER");
        }
        TokenBalance storage bal = nodeTokenBalance[req.node][token];
        uint numCanWithdraw = FullMath.min(bal.numOnWithdraw, req.amount);
        bal.numOnWithdraw = numCanWithdraw <= req.amount ? 0 : uint(bal.numOnWithdraw).sub(req.amoun
t).toUint128();
        req.status = 2;
        TransferHelper.safeTransfer2(token, req.node, numCanWithdraw);
        emit AssetsClaimed(requestID, req.node, numCanWithdraw, token);
    }
```

danielt : Use the Checks-Effects-Interactions best practice and make all state changes before calling external contracts. Also, consider using function modifiers such as `nonReentrant` from Reentrancy Guard to prevent re-entrancy at the contract level.
Yaodao : Recommend using the Checks-Effects-Interactions pattern to avoid reentrancy attack. Also, consider using function modifiers such as `nonReentrant` from Reentrancy Guard to prevent re-entrancy at the contract level.
Consider below fix in the function

```
    bal.numOnWithdraw = numCanWithdraw <= req.amount ? 0 : uint(bal.numOnWithdraw).sub(req.amount).t
oUint128();
    req.status = 2;
    TransferHelper.safeTransfer2(token, req.node, numCanWithdraw);
```

0xzoobi : Use the Checks-Effects-Interactions best practice and update the variables before making external calls. Also, consider using function modifiers such as `nonReentrant` from Reentrancy Guard to prevent re-entrancy at the contract level.

Client Response

Fixed

DAP-11:Use TransferHelper library to transfer ERC20.

Category	Severity	Status	Contributor
Logical	Medium	Fixed	jayphbee, 0xzoobi

Code Reference

- code/contracts/peripherals/common/ERC20Service.sol#L27
- code/contracts/peripherals/common/ERC20Service.sol#L34

```
27:             IERC20(token).approve(spender, amount);

34:             IERC20(token).transfer(to, amount);
```

Description

jayphbee : There are token that doesn't follow the standard ERC20 implementation like USDT in the ethereum mainnet. It will revert when calling the standard ERC20 `transfer` interface.
0xzoobi : If the `transfer` call returns false instead of revert, the ERC20Service's `execute` function will end successfully but no transfer of tokens will have taken place.
The token address is also an user controlled input hence, non-openzeppelin standard ERC20 tokens can also be part of it.

Recommendation

jayphbee : User `TransferHelper` to transfer ERC20

```
        } else if (p.action == 1) {
            // Transfer
            (address token, address to, uint256 amount) = abi.decode(
                data[32:],
                (address, address, uint256)
            );
            TransferHelper.safeTransfer(token, to, amount); // Use safeTrasnfer
        }
```

0xzoobi : check for boolean return value of transfer or use `safeERC20Transfer` via `SafeERC20.sol`

Client Response

Fixed

DAP-12: `_feeReceiver` is user controlled in the payFee function

Category	Severity	Status	Contributor
Logical	Medium	Acknowledged	0xzoobi

Code Reference

- code/contracts/core/VirtualWallet.sol#L52-L67

```
52:     function payFee(
53:         address _feeToken,
54:         uint256 _gasFee,
55:         address _feeReceiver
56:     ) external override onlyVWManager returns (bool success) {
57:         if (_feeToken == address(0)) {
58:             //  TransferHelper.safeTransferETH(_feeReceiver, _gasFee);
59:             (success, ) = _feeReceiver.call{value: _gasFee}("");
60:         } else {
61:             //  TransferHelper.safeTransfer(_feeToken, _feeReceiver, _gasFee);
62:             (bool res, bytes memory data) = _feeToken.call(
63:                 abi.encodeWithSelector(0xa9059cbb, _feeReceiver, _gasFee)
64:             );
65:             success = res && (data.length == 0 || abi.decode(data, (bool)));
66:         }
67:     }
```

Description

0xzoobi : `_feeReceiver` is the address that the `onlyVWManager` decides when calling the function. Currently there are multiple issues with the implementation.

First, there is a zero-address check missing for `_feeReceiver`. Secondly, there is no check who the `_feeReceiver` actually is. The inputs can be controlled by user since the function is marked `external`.

Recommendation

0xzoobi : Add proper checks to check if the `_feeReceiver` and mark the function as `internal` if you do not want the user directly calling the function.

Client Response

Acknowledged.We design to allown executor to set feereceiver cause executor pay for the gas

DAP-13: `_wallet` and `realWallet` doesn't implement the `IVirtualWallet` interface

Category	Severity	Status	Contributor
Logical	Medium	Acknowledged	jayphbee

Code Reference

- code/contracts/core/PayDB.sol#L190-L195
- code/contracts/core/PayDB.sol#L441-L446

```
190:                 IVirtualWallet(realWallet).execute(
191:                     eparam.code,
192:                     eparam.service,
193:                     data,
194:                     serviceSignature
195:                 );

441:                         IVirtualWallet(_wallet).execute(
442:                             eparam[i].code,
443:                             eparam[i].service,
444:                             data,
445:                             serviceSignature
446:                         );
```

Description

jayphbee : `_wallet` and `realWallet` doesn't implement the `IVirtualWallet` interface thus when `eparam.manager == address(0)`, `_executeDstOrder` and `_executeIsolateOrder` will always revert. `VirtualWallet` contract implements the `IVirtualWalletV2` interface instead of the `IVirtualWallet` interface.

```
if (eparam.manager == address(0)) {
                IVirtualWallet(realWallet).execute(
                    eparam.code,
                    eparam.service,
                    data,
                    serviceSignature
                );
            }
```

Recommendation

jayphbee : User the `IVritaulWalletV2` interface.

```
IVirtualWalletV2(_wallet).execute(
                            eparam[i].code,
                            eparam[i].service,
                            data
                        );

IVirtualWalletV2(realWallet).execute(
                    eparam.code,
                    eparam.service,
                    data
                );
```

Client Response

Acknowledged.we are going to abond the old version virtual wallet

DAP-14:for loop is early return in the `award` function

Category	Severity	Status	Contributor
Logical	Medium	Fixed	jayphbee, danielt, Yaodao

Code Reference

- code/contracts/peripherals/reward/RewardPool.sol#L25-L69
- code/contracts/peripherals/reward/RewardPool.sol#L37-L68

```
25:     function award(
26:         address[] calldata tokens,
27:         uint256[] calldata amounts,
28:         address to
29:     ) external override {
30:         require(rewardConfig.service != address(0), "NO REWARD CONFIG");
31:         require(
32:             IVWManager(rewardConfig.gateway).volatileService() ==
33:                 rewardConfig.service,
34:             "UNAUTHORIZED SERVICE"
35:         );
36:         require(tokens.length == amounts.length, "TOKE AMOUNT NO MATCH");
37:         for (uint256 i = 0; i < tokens.length; i++) {
38:             require(claimConfigs[tokens[i]].k > 0, "K NOT SET");
39:             if (rewardConfig.rewardToken != address(0)) {
40:                 TransferHelper.safeTransfer2(
41:                     rewardConfig.rewardToken,
42:                     to,
43:                     FullMath.min(
44:                         FullMath.mulDiv(
45:                             amounts[i],
46:                             claimConfigs[tokens[i]].k,
47:                             denonimator
48:                         ),
49:                         IERC20(rewardConfig.rewardToken).balanceOf(
50:                             address(this)
51:                         )
52:                     )
53:                 );
54:                 return;
55:             }
56:             TransferHelper.safeTransfer2(
57:                 rewardConfig.rewardToken,
58:                 to,
59:                 FullMath.min(
60:                     FullMath.mulDiv(
61:                         amounts[i],
62:                         claimConfigs[tokens[i]].k,
63:                         denonimator
64:                     ),
65:                     address(this).balance
66:                 )
```

```
67:             );
68:         }
69:     }

37:         for (uint256 i = 0; i < tokens.length; i++) {
38:             require(claimConfigs[tokens[i]].k > 0, "K NOT SET");
39:             if (rewardConfig.rewardToken != address(0)) {
40:                 TransferHelper.safeTransfer2(
41:                     rewardConfig.rewardToken,
42:                     to,
43:                     FullMath.min(
44:                         FullMath.mulDiv(
45:                             amounts[i],
46:                             claimConfigs[tokens[i]].k,
47:                             denonimator
48:                         ),
49:                         IERC20(rewardConfig.rewardToken).balanceOf(
50:                             address(this)
51:                         )
52:                     )
53:                 );
54:                 return;
55:             }
56:             TransferHelper.safeTransfer2(
57:                 rewardConfig.rewardToken,
58:                 to,
59:                 FullMath.min(
60:                     FullMath.mulDiv(
61:                         amounts[i],
62:                         claimConfigs[tokens[i]].k,
63:                         denonimator
64:                     ),
65:                     address(this).balance
66:                 )
67:             );
68:         }
```

Description

jayphbee : If `rewardConfig.rewardToken != address(0)` the for loop is early return in the `reward` function.

```
for (uint256 i = 0; i < tokens.length; i++) {
            require(claimConfigs[tokens[i]].k > 0, "K NOT SET");
            if (rewardConfig.rewardToken != address(0)) {
                TransferHelper.safeTransfer2(
                    rewardConfig.rewardToken,
                    to,
                    FullMath.min(
                        FullMath.mulDiv(
                            amounts[i],
                            claimConfigs[tokens[i]].k,
                            denonimator
                        ),
                        IERC20(rewardConfig.rewardToken).balanceOf(
                            address(this)
                        )
                    )
                );
                return; // @audit should not early return
            }
            TransferHelper.safeTransfer2(
                rewardConfig.rewardToken,
                to,
                FullMath.min(
                    FullMath.mulDiv(
                        amounts[i],
                        claimConfigs[tokens[i]].k,
                        denonimator
                    ),
                    address(this).balance
                )
            );
        }
```

This is not the intended behavior. The impact is that `to` address could receive less funds than expected.

danielt : In the `RewardPool` contract, the `award` function will distribute rewards to an account. However, the `rewardToken` will be distributed to the same account twice if `rewardConfig.rewardToken != address(0)`, as shown below:

```
        for (uint256 i = 0; i < tokens.length; i++) {
            require(claimConfigs[tokens[i]].k > 0, "K NOT SET");
            if (rewardConfig.rewardToken != address(0)) {
                TransferHelper.safeTransfer2(
                    rewardConfig.rewardToken,
                    to,
                    FullMath.min(
                        FullMath.mulDiv(
                            amounts[i],
                            claimConfigs[tokens[i]].k,
                            denonimator
                        ),
                        IERC20(rewardConfig.rewardToken).balanceOf(
                            address(this)
                        )
                    )
                );
                return;
            }
            TransferHelper.safeTransfer2(
                rewardConfig.rewardToken,
                to,
                FullMath.min(
                    FullMath.mulDiv(
                        amounts[i],
                        claimConfigs[tokens[i]].k,
                        denonimator
                    ),
                    address(this).balance
                )
            );
        }
```

Note that, if `rewardConfig.rewardToken != address(0)`, the amount of the first transfer to the user will be an amount of `IERC20(rewardConfig.rewardToken).balanceOf(address(this))`, and the amount of the second transfer to the user will be an amount of `address(this).balance`.

As some results:

- The number of the sum of `IERC20(rewardConfig.rewardToken).balanceOf(address(this))` and `address(this).balance`, will be distributed to the user, which will incur reward token loss and potentially revert if there is no enough reward token in the contract.
- if the actual number of reward tokens is less than the amount of `address(this).balance`, the transaction will revert.

  Yaodao : According to the following codes, the use of `if (rewardConfig.rewardToken != address(0))` in the loop is used to deal with the different transfers between platform tokens and other tokens. The `award()` function

is used to transfer the given amounts of the given tokens to the `to` address. However, the keyword `return` is used and it will exit the loop in the logic of the case is true. As a result, the rest tokens will not be transferred to the `to` address. It seems that the keyword `continue` should be used instead of the keyword `return`.

Consider below codes

```
    function award(
        address[] calldata tokens,
        uint256[] calldata amounts,
        address to
    ) external override {
        ...
        for (uint256 i = 0; i < tokens.length; i++) {
            require(claimConfigs[tokens[i]].k > 0, "K NOT SET");
            if (rewardConfig.rewardToken != address(0)) {
                TransferHelper.safeTransfer2(
                    rewardConfig.rewardToken,
                    to,
                    FullMath.min(
                        FullMath.mulDiv(
                            amounts[i],
                            claimConfigs[tokens[i]].k,
                            denonimator
                        ),
                        IERC20(rewardConfig.rewardToken).balanceOf(
                            address(this)
                        )
                    )
                );
                return;
            }
            TransferHelper.safeTransfer2(
                rewardConfig.rewardToken,
                to,
                FullMath.min(
                    FullMath.mulDiv(
                        amounts[i],
                        claimConfigs[tokens[i]].k,
                        denonimator
                    ),
                    address(this).balance
                )
            );
        }
    }
```

Recommendation

jayphbee : Remove the `return` statement and introduce the `else` branch.

```
for (uint256 i = 0; i < tokens.length; i++) {
            require(claimConfigs[tokens[i]].k > 0, "K NOT SET");
            if (rewardConfig.rewardToken != address(0)) {
                TransferHelper.safeTransfer2(
                    rewardConfig.rewardToken,
                    to,
                    FullMath.min(
                        FullMath.mulDiv(
                            amounts[i],
                            claimConfigs[tokens[i]].k,
                            denonimator
                        ),
                        IERC20(rewardConfig.rewardToken).balanceOf(
                            address(this)
                        )
                    )
                );
            } else {
                TransferHelper.safeTransfer2(
                    rewardConfig.rewardToken,
                    to,
                    FullMath.min(
                        FullMath.mulDiv(
                            amounts[i],
                            claimConfigs[tokens[i]].k,
                            denonimator
                        ),
                        address(this).balance
                    )
                );
            }
```

danielt : Making the second transfer of the reward token into an `else` branch for only transferring the reward token when the reward token is `address(0)`.
Yaodao : Recommend using `continue` to replace the `return`.

Client Response

Fixed

DAP-15:Incorrect check in the function `submitWithdrawRequest()`

Category	Severity	Status	Contributor
Logical	Low	Fixed	Yaodao

Code Reference

- code/contracts/governance/PayLock.sol#L94-L109

```
94:    function submitWithdrawRequest(address token, uint amount, address node) external returns (ui
nt requestID){
95:        if (msg.sender != node) {
96:            require(Ownable(node).owner() == msg.sender, "ONLY NODE/NODE_OWNER");
97:        }
98:        TokenBalance storage bal = nodeTokenBalance[node][token];
99:        require(bal.numTotal >= bal.numOnWithdraw + amount, "INSUFFICIENT_AMOUNT");
100:        bal.numOnWithdraw = uint(bal.numOnWithdraw).add(amount).toUint128();
101:        bal.numTotal = uint(bal.numTotal).sub(amount).toUint128();
102:
103:        requestID = numWithdrawRequest++;
104:        WithdrawRequest storage req = withdrawRequests[requestID];
105:        (req.token, req.submitTime, req.status, req.amount, req.node) = (
106:        token, uint32(block.timestamp), uint32(1), amount.toUint128(), node
107:        );
108:        emit WithdrawRequestSubmitted(requestID, node, amount, token);
109:    }
```

Description

Yaodao : The function `submitWithdrawRequest()` is used to submit the request to withdraw certain assets. The check `require(bal.numTotal >= bal.numOnWithdraw + amount, "INSUFFICIENT_AMOUNT");` is used to ensure the amount to request is enough. However, the `bal.numTotal` and `bal.numOnWithdraw` are updated synchronizedly. As a result, the check will be incorrect before the service node claim this request and the `bal.numOnWithdraw` is updated back in the call of `claim()`.

Consider below codes

```
    function submitWithdrawRequest(address token, uint amount, address node) external returns (uint
requestID){
        if (msg.sender != node) {
            require(Ownable(node).owner() == msg.sender, "ONLY NODE/NODE_OWNER");
        }
        TokenBalance storage bal = nodeTokenBalance[node][token];
        require(bal.numTotal >= bal.numOnWithdraw + amount, "INSUFFICIENT_AMOUNT");
        bal.numOnWithdraw = uint(bal.numOnWithdraw).add(amount).toUint128();
        bal.numTotal = uint(bal.numTotal).sub(amount).toUint128();

        requestID = numWithdrawRequest++;
        WithdrawRequest storage req = withdrawRequests[requestID];
        (req.token, req.submitTime, req.status, req.amount, req.node) = (
        token, uint32(block.timestamp), uint32(1), amount.toUint128(), node
        );
        emit WithdrawRequestSubmitted(requestID, node, amount, token);
    }
```

For example, assuming the service node `A` deposit 1000 tokenA.

Then the service node `A` `submitWithdrawRequest` for 600 tokenA. The check is `1000 >= 0 + 600`. The `bal.numOnWithdraw` will be 600 and the `bal.numTotal` will be 400.

For now, the service node `A` should be able to submit another request for 400 tokenA. However, if the service node `A` call this submit. The check will be `400 >= 600 + 400` and the call will fail. Unless the service node `A` call the `claim()` first to make the `bal.numOnWithdraw` to be 0 again, which is not suitable.

Since `bal.numTotal` is updated when it is submitted, its value is already subtracted from the amount of pending tokens to be claimed(`bal.numOnWithdraw`). So it should not be compared with the sum of the `bal.numOnWithdraw` and the amount given in the submit.

Recommendation

Yaodao : Recommend updating the check as following.

Consider below fix in the function

```
    require(bal.numTotal >= amount, "INSUFFICIENT_AMOUNT");
```

Client Response

Fixed

DAP-16:Incorrect check of deflationary token

Category	Severity	Status	Contributor
Logical	Low	Fixed	Yaodao

Code Reference

- code/contracts/libraries/TransferHelper.sol#L44

```
44:         require(balanceBefore <= balanceAfter, 'No deflationary token');
```

Description

Yaodao : According to the error message, the check `require(balanceBefore <= balanceAfter, 'No deflationary token');` is used to check whether the token is not a deflationary token. However, the check only compares the balances of the `to` address before and after the transfer which can't check whether the token is not a deflationary token.

Consider below codes

```
    function safeTransferFrom3(
        address token,
        address from,
        address to,
        uint256 value
    ) internal returns (uint){
        uint256 balanceBefore = IERC20(token).balanceOf(to);

        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to,
value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER
_FROM_FAILED');
        uint256 balanceAfter = IERC20(token).balanceOf(to);
        require(balanceBefore <= balanceAfter, 'No deflationary token');
        return (balanceAfter - balanceBefore);
    }
```

Recommendation

Yaodao : Recommend updating the check.

Consider below fix in the `safeTransferFrom3()` function

```
    require(balanceBefore + value <= balanceAfter, 'No deflationary token');
```

Client Response

Fixed

DAP-17:Lack array parameters length equality check

Category	Severity	Status	Contributor
Logical	**Low**	**Fixed**	**jayphbee, Yaodao**

Code Reference

- code/contracts/governance/PayLock.sol#L129
- code/contracts/core/PayDB.sol#L407-L476

```
129:    function punish(

407:    function _executeIsolateOrder(
408:        ExeOrderParam[] calldata eparam,
409:        bytes calldata data,
410:        bytes calldata serviceSignature
411:    ) internal {
412:        (address _receiver, address _wallet) = (
413:            eparam[0].receiver,
414:            eparam[0].wallet
415:        );
416:        if (_receiver == address(0)) {
417:            // Wallet not exists.
418:            // In this case, both wallet and receiver will be the newly created wallet.
419:            address _VWManager = (eparam[0].manager == address(0))
420:                ? VWManager
421:                : eparam[0].manager;
422:            _receiver = createWalletIfNotExists(_VWManager, eparam[0].wallet);
423:            _wallet = _receiver;
424:        }
425:        uint256 totalETH;
426:        for (uint256 i = 0; i < eparam.length; i++) {
427:            if (eparam[i].tokenOut == address(0)) {
428:                TransferHelper.safeTransferETH(_receiver, eparam[i].amountOut);
429:                totalETH += eparam[i].amountOut;
430:            } else {
431:                TransferHelper.safeTransferFrom(
432:                    eparam[i].tokenOut,
433:                    msg.sender,
434:                    _receiver,
435:                    eparam[i].amountOut
436:                );
437:            }
438:            if (i == eparam.length - 1) {
439:                if (eparam[i].code != 0 && serviceSignature.length > 0) {
440:                    if (eparam[i].manager == address(0)) {
441:                        IVirtualWallet(_wallet).execute(
442:                            eparam[i].code,
443:                            eparam[i].service,
444:                            data,
445:                            serviceSignature
```

```
446:                         );
447:                     } else {
448:                         IVWManager(eparam[i].manager).execute(
449:                             IVWManager.ExecuteParam({
450:                                 wallet: _wallet,
451:                                 code: eparam[i].code,
452:                                 service: eparam[i].service,
453:                                 data: data,
454:                                 proof: eparam[i].proof,
455:                                 payToken: eparam[i].payToken,
456:                                 gasTokenPrice: eparam[i].gasTokenPrice,
457:                                 priorityFee: eparam[i].priorityFee,
458:                                 gasLimit: eparam[i].gasLimit
459:                             }),
460:                             serviceSignature,
461:                             eparam[i].isGateway,
462:                             eparam[i].feeReceiver
463:                         );
464:                     }
465:                 }
466:             }
467:
468:             emit IsolateOrderExecuted(
469:                 msg.sender,
470:                 eparam[i].amountOut,
471:                 eparam[i].tokenOut,
472:                 eparam[i].receiver
473:             );
474:         }
475:         require(msg.value == totalETH);
476:     }
```

Description

jayphbee : There is no array length equality check for `tokens` and `amount` in the punish function. If `tokens.length > amounts.length` `punish` will revert due to index out of bounds. If `tokens.length < amounts.length` , it means that some token will not be slashed

Yaodao : In the function `_executeIsolateOrder()`, the parameter `eparam` is an array given by the caller. the value of `_receiver` and `_wallet` uses the value of the index 0 in the array `eparam` (Create new wallet when the `_receiver` is address(0)). However, the other parameters used in the call of `IVirtualWallet(_wallet).execute()` or `IV-`

`WManager(eparam[i].manager).execute()` uses the value of the index `eparam.length - 1` in the array `epa ram`. As a result, the parameters used in the `excute()` may mismatch.

Consider below codes

```
    function _executeIsolateOrder(
        ExeOrderParam[] calldata eparam,
        bytes calldata data,
        bytes calldata serviceSignature
    ) internal {
        (address _receiver, address _wallet) = (
            eparam[0].receiver,
            eparam[0].wallet
        );
        if (_receiver == address(0)) {
            // Wallet not exists.
            // In this case, both wallet and receiver will be the newly created wallet.
            address _VWManager = (eparam[0].manager == address(0))
                ? VWManager
                : eparam[0].manager;
            _receiver = createWalletIfNotExists(_VWManager, eparam[0].wallet);
            _wallet = _receiver;
        }
        uint256 totalETH;
        for (uint256 i = 0; i < eparam.length; i++) {
            ...
            if (i == eparam.length - 1) {
                if (eparam[i].code != 0 && serviceSignature.length > 0) {
                    if (eparam[i].manager == address(0)) {
                        IVirtualWallet(_wallet).execute(
                            eparam[i].code,
                            eparam[i].service,
                            data,
                            serviceSignature
                        );
                    } else {
                        IVWManager(eparam[i].manager).execute(
                            IVWManager.ExecuteParam({
                                wallet: _wallet,
                                code: eparam[i].code,
                                service: eparam[i].service,
                                data: data,
                                proof: eparam[i].proof,
                                payToken: eparam[i].payToken,
                                gasTokenPrice: eparam[i].gasTokenPrice,
                                priorityFee: eparam[i].priorityFee,
                                gasLimit: eparam[i].gasLimit
                            }),
```

```
    serviceSignature,
                                eparam[i].isGateway,
                                eparam[i].feeReceiver
                            );
                        }
                    }
                }
                ...
    }
```

Recommendation

jayphbee : Add array length equality check for `tokens` and `amounts`.

```
function punish(
        uint orderId,
        address node,
        address to,
        address[] calldata tokens,
        uint[] calldata amounts
    )
    external onlyOwner {
        requie(tokens.length > 0 && tokens.length == amounts.length, "Invalid length");
        ...
    }
```

Yaodao : Recommend using the corresponding params or stating for the logic.

Client Response

Fixed

DAP-18:Lack of checking result of the `ECDSA.recover` function

Category	Severity	Status	Contributor
Signature Forgery or Replay	Low	Fixed	danielt

Code Reference

- code/contracts/libraries/SignLibrary.sol#L52-L62

```
52:     function verify(
53:         address owner,
54:         bytes32 domain,
55:         bytes32 dataHash,
56:         bytes memory signature
57:     ) external pure {
58:         bytes32 digest = keccak256(
59:             abi.encodePacked("\x19\x01", domain, dataHash)
60:         );
61:         require(owner == ECDSA.recover(digest, signature), "E1");
62:     }
```

Description

danielt : The solidity function `ecrecover` in the `SignLibrary` is used in contracts of this project, however, the error result of 0 is not checked for. As the documentation shown below: https://docs.soliditylang.org/en/v0.8.9/units-and-global-variables.html?highlight=ecrecover#mathematical-and-cryptographic-functions "recover the address associated with the public key from the elliptic curve signature or return zero on error. "

It is always a good practice to check the error result of 0 for the `ecrecover` to prevent potential unexpected result.

Recommendation

danielt : Check the error result of 0 for the `ecrecover` to prevent potential unexpected result.

Client Response

Fixed

DAP-19:Lack of checks and updates to `result[wallet][codeToCancel]`

Category	Severity	Status	Contributor
Logical	Low	Fixed	Yaodao

Code Reference

- code/contracts/core/vwmanager/VWManagerService.sol#L27-L60

```
27:     function cancelTx(
28:         uint256 code,
29:         address wallet,
30:         uint256 codeToCancel,
31:         FeeParam calldata fParam,
32:         bytes calldata signature
33:     ) external {
34:         uint256 preGas = gasleft();
35:         require(result[walletOwner[wallet]][code] == 0, "E2");
36:         (uint256 dstChainId, uint256 srcChain, uint256 time) = VWCode
37:             .chainidsAndExpTime(code);
38:         require(dstChainId == block.chainid && block.timestamp < time, "E3OrE6");
39:
40:         bytes32 dataHash = keccak256(
41:             abi.encode(
42:                 CANCEL_TYPEHASH,
43:                 code,
44:                 codeToCancel,
45:                 fParam.payToken,
46:                 fParam.gasTokenPrice,
47:                 fParam.priorityFee
48:             )
49:         );
50:
51:         result[walletOwner[wallet]][code] = 3;
52:
53:         SignLibrary.verify(
54:             walletOwner[wallet],
55:             domainSeparator[srcChain],
56:             dataHash,
57:             signature
58:         );
59:         _walletPayFee(wallet, preGas, fParam);
60:     }
```

Description

Yaodao : In the function `cancelTx()`, the parameter `codeToCancel` is used to check the signature but the value of `result[wallet][codeToCancel]` is not updated. According to the name of the function, the function `cancelTx()` is used to cancel the transaction. In other functions in this contract, the `result[wallet][code]` will be compared with `0`. Other functions can still be called after the call of function `cancelTx()` because the value of `result[walle-`

`t][codeToCancel]` is not updated. Besides, the value of `result[wallet][codeToCancel]` is not checked at the beginning. As a result, whether the transaction is executed or can be canceled is not checked.

Consider below codes

```
    function cancelTx(
        uint256 code,
        address wallet,
        uint256 codeToCancel,
        FeeParam calldata fParam,
        bytes calldata signature
    ) external {
        uint256 preGas = gasleft();
        require(result[walletOwner[wallet]][code] == 0, "E2");
        (uint256 dstChainId, uint256 srcChain, uint256 time) = VWCode
            .chainidsAndExpTime(code);
        require(dstChainId == block.chainid && block.timestamp < time, "E3OrE6");

        bytes32 dataHash = keccak256(
            abi.encode(
                CANCEL_TYPEHASH,
                code,
                codeToCancel,
                fParam.payToken,
                fParam.gasTokenPrice,
                fParam.priorityFee
            )
        );

        result[walletOwner[wallet]][code] = 3;

        SignLibrary.verify(
            walletOwner[wallet],
            domainSeparator[srcChain],
            dataHash,
            signature
        );
        _walletPayFee(wallet, preGas, fParam);
    }
```

Recommendation

Yaodao : Recommend adding the logic to check and update the value of `result[wallet][codeToCancel]`.

Client Response

Fixed

DAP-20:Lack of repeatability check

Category	Severity	Status	Contributor
Logical	Low	Fixed	Yaodao

Code Reference

- code/contracts/core/vwmanager/VWManagersReader.sol#L22-L25
- code/contracts/core/vwmanager/VWManagersReader.sol#L27-L34

```
22:     function addVWManager(address _VWManager) public override onlyOwner {
23:         VWManagers.push(_VWManager);
24:         emit AddVWManager(_VWManager);
25:     }

27:     function updateVWManager(uint256 index, address _VWManager)
28:         public
29:         override
30:         onlyOwner
31:     {
32:         VWManagers[index] = _VWManager;
33:         emit UpdateVWManager(index, _VWManager);
34:     }
```

Description

Yaodao : In the function `addVWManager()`, the manager is added directly without checking whether the manager exists. And in the function `updateVWManager()`, the given index manager is updated to a new manager directly without checking whether the new manager exists in the other index. Besides, there is no function to remove the duplicate managers in the array `VWManagers` although the `updateVWManager()` can update a duplicate manager to a new manager or `address(0)` which is not a suitable way.

Consider below codes

```
    function addVWManager(address _VWManager) public override onlyOwner {
        VWManagers.push(_VWManager);
        emit AddVWManager(_VWManager);
    }

    function updateVWManager(uint256 index, address _VWManager)
        public
        override
        onlyOwner
    {
        VWManagers[index] = _VWManager;
        emit UpdateVWManager(index, _VWManager);
    }
```

Recommendation

Yaodao : Recommend adding the repeatability check to avoid adding duplicate manager.

Client Response

Fixed

DAP-21:Potential Reentrancy risk in `VirtualWallet` contract `payFee` function

Category	Severity	Status	Contributor
Reentrancy	Low	Fixed	hunya

Code Reference

- code/contracts/core/VirtualWallet.sol#L52-L67

```
52:     function payFee(
53:         address _feeToken,
54:         uint256 _gasFee,
55:         address _feeReceiver
56:     ) external override onlyVWManager returns (bool success) {
57:         if (_feeToken == address(0)) {
58:             //  TransferHelper.safeTransferETH(_feeReceiver, _gasFee);
59:             (success, ) = _feeReceiver.call{value: _gasFee}("");
60:         } else {
61:             //  TransferHelper.safeTransfer(_feeToken, _feeReceiver, _gasFee);
62:             (bool res, bytes memory data) = _feeToken.call(
63:                 abi.encodeWithSelector(0xa9059cbb, _feeReceiver, _gasFee)
64:             );
65:             success = res && (data.length == 0 || abi.decode(data, (bool)));
66:         }
67:     }
```

Description

hunya : Function `payFee` in the VirtualWallet contract being vulnerable to a reentrancy attack. Function `payFee` uses the underlying function `call` with unlimited gas and would call the fallback function, and the address parameter `_feeReceiver` is passed in from the outside.

Recommendation

hunya : We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts. Also, consider using function modifiers such as `nonReentrant` from Reentrancy Guard to prevent re-entrancy at the contract level.

Client Response

Fixed

DAP-22:Return value not checked.

Category	Severity	Status	Contributor
Logical	Low	Fixed	jayphbee, Yaodao, 0xzoobi

Code Reference

- code/contracts/peripherals/common/ERC20Service.sol#L34
- code/contracts/core/PayDB.sol#L190
- code/contracts/core/vwmanager/VWManager.sol#L201
- code/contracts/core/PayDB.sol#L210
- code/contracts/core/PayDB.sol#L441
- code/contracts/core/PayDB.sol#L448

```
34:             IERC20(token).transfer(to, amount);

190:                 IVirtualWallet(realWallet).execute(

201:              splitAndSendFee(

210:                 IVWManager(eparam.manager).execute(

441:                         IVirtualWallet(_wallet).execute(

448:                         IVWManager(eparam[i].manager).execute(
```

Description

jayphbee : if `protocolFeeOpened` is true, `splitAndSendFee` will be called, but it's return value is sliently ignored. `splitAndSendFee` calls `payFee` to transfer ether or ERC20 token, but not revert on error. The caller has the responsiablity to validate the return value, otherwise the protocol fee is not always guaranteed to be received if the `_feeToken` is an ERC20 that returning false when `transfer` failed.

```
    function payFee(
        address _feeToken,
        uint256 _gasFee,
        address _feeReceiver
    ) external override onlyVWManager returns (bool success) {
        if (_feeToken == address(0)) {
            //  TransferHelper.safeTransferETH(_feeReceiver, _gasFee);
            (success, ) = _feeReceiver.call{value: _gasFee}("");
        } else {
            //  TransferHelper.safeTransfer(_feeToken, _feeReceiver, _gasFee);
            (bool res, bytes memory data) = _feeToken.call(
                abi.encodeWithSelector(0xa9059cbb, _feeReceiver, _gasFee)
            );
            success = res && (data.length == 0 || abi.decode(data, (bool))); // @audit _feeToken can
return false when transfer failed
        }
    }
```

Yaodao : The following codes in the function `execute()` not check the return value of `IERC20(token).transfer`.

Consider below codes

```
        // Transfer
        (address token, address to, uint256 amount) = abi.decode(
            data[32:],
            (address, address, uint256)
        );
        IERC20(token).transfer(to, amount);
```

0xzoobi : The interface for `execute` function expects a Boolean return value to check the status of the execute call but the actual implementation does not check for the return value.

The impact of this being, the function retuned a false value, but since the return value was never checked, it was assumed to be executed successfully.

Recommendation

jayphbee : check the return value of `splitAndSendFee` function.

```
        if (protocolFeeOpened) {
            res = splitAndSendFee( // explicitly handle the return value
                eParam.wallet,
                eParam.payToken,
                gasFee,
                feeReceiver
            );
        } else {
            res = IVirtualWalletV2(eParam.wallet).payFee(
                eParam.payToken,
                gasFee,
                feeReceiver
            );
        }
        require(res, "E10");
```

Yaodao : Recommend checking the return value of the `transfer()`.
0xzoobi : check for Boolean return value whenever `execute` is invoked.

Client Response

Fixed

DAP-23:Uncheck the result of the `VirtualWallet.execute()`

Category	Severity	Status	Contributor
Code Style	Low	Fixed	Yaodao

Code Reference

- code/contracts/core/VirtualWallet.sol#L37-L50
- code/contracts/core/PayDB.sol#L168-L237

```
37:     function execute(
38:         uint256 code,
39:         address service,
40:         bytes calldata data
41:     ) external override onlyVWManager returns (bool res) {
42:         (res, ) = service.delegatecall(
43:             abi.encodeWithSelector(
44:                 0x57c97782, //IService.execute.selector,
45:                 code,
46:                 data,
47:                 msg.sender
48:             )
49:         );
50:     }

168:     function _executeDstOrder(
169:         ExeOrderParam calldata eparam,
170:         address realWallet,
171:         bytes memory data,
172:         bytes memory serviceSignature
173:     ) internal {
174:         (uint256 dstChainId, , uint256 time) = OrderId.chainidsAndExpTime(
175:             eparam.payOrderId
176:         );
177:
178:         require(block.chainid == dstChainId && block.timestamp < time, "E8");
179:
180:         bytes32 workFlowHash = keccak256(
181:             abi.encode(
182:                 eparam.wallet,
183:                 eparam.code,
184:                 eparam.service,
185:                 keccak256(data)
186:             )
187:         );
188:         if (eparam.code != 0 && serviceSignature.length > 0) {
189:             if (eparam.manager == address(0)) {
190:                 IVirtualWallet(realWallet).execute(
191:                     eparam.code,
192:                     eparam.service,
193:                     data,
194:                     serviceSignature
```

```
195:                 );
196:             } else {
197:                 IVWManager.ExecuteParam memory exeParam = IVWManager
198:                     .ExecuteParam({
199:                         wallet: realWallet,
200:                         code: eparam.code,
201:                         service: eparam.service,
202:                         data: data,
203:                         proof: eparam.proof,
204:                         payToken: eparam.payToken,
205:                         gasTokenPrice: eparam.gasTokenPrice,
206:                         priorityFee: eparam.priorityFee,
207:                         gasLimit: eparam.gasLimit
208:                     });
209:
210:                 IVWManager(eparam.manager).execute(
211:                     exeParam,
212:                     serviceSignature,
213:                     eparam.isGateway,
214:                     eparam.feeReceiver
215:                 );
216:             }
217:         }
218:
219:         require(dstOrder[eparam.payOrderId] == 0, "E7");
220:         dstOrder[eparam.payOrderId] = keccak256(
221:             abi.encode(
222:                 eparam.amountOut,
223:                 eparam.tokenOut,
224:                 eparam.receiver,
225:                 workFlowHash
226:             )
227:         );
228:
229:         emit OrderExecuted(
230:             msg.sender,
231:             eparam.amountOut,
232:             eparam.tokenOut,
233:             eparam.receiver,
234:             eparam.payOrderId,
235:             workFlowHash
236:         );
237:     }
```

Description

Yaodao : In the function `_executeDstOrder`, `VirtualWallet.execute()` is called but not check the result of the call. Besides, in the function `VirtualWallet.execute()`, the result of the `service.delegatecall()` is also called but not check the result.

Consider below codes

```
    function _executeDstOrder(
        ExeOrderParam calldata eparam,
        address realWallet,
        bytes memory data,
        bytes memory serviceSignature
    ) internal {
        (uint256 dstChainId, , uint256 time) = OrderId.chainidsAndExpTime(
            eparam.payOrderId
        );

        require(block.chainid == dstChainId && block.timestamp < time, "E8");

        bytes32 workFlowHash = keccak256(
            abi.encode(
                eparam.wallet,
                eparam.code,
                eparam.service,
                keccak256(data)
            )
        );
        if (eparam.code != 0 && serviceSignature.length > 0) {
            if (eparam.manager == address(0)) {
                IVirtualWallet(realWallet).execute(
                    eparam.code,
                    eparam.service,
                    data,
                    serviceSignature
                );
            } else {
                IVWManager.ExecuteParam memory exeParam = IVWManager
                    .ExecuteParam({
                        wallet: realWallet,
                        code: eparam.code,
                        service: eparam.service,
                        data: data,
                        proof: eparam.proof,
                        payToken: eparam.payToken,
                        gasTokenPrice: eparam.gasTokenPrice,
                        priorityFee: eparam.priorityFee,
                        gasLimit: eparam.gasLimit
                    });

                IVWManager(eparam.manager).execute(
```

```
exeParam,
                    serviceSignature,
                    eparam.isGateway,
                    eparam.feeReceiver
                );
            }
        }

        require(dstOrder[eparam.payOrderId] == 0, "E7");
        dstOrder[eparam.payOrderId] = keccak256(
            abi.encode(
                eparam.amountOut,
                eparam.tokenOut,
                eparam.receiver,
                workFlowHash
            )
        );
```

Recommendation

Yaodao : Recommend checking the result of the call of `VirtualWallet.execute()`.

Client Response

Fixed

DAP-24:VirtaulWallet should have the ability to receive ERC1155.

Category	Severity	Status	Contributor
Logical	Low	Fixed	jayphbee, 0xzoobi

Code Reference

- code/contracts/core/VirtualWallet.sol#L79
- code/contracts/core/VirtualWallet.sol#L79-L87

```
79:    function onERC721Received(

79:    function onERC721Received(
80:        address,
81:        address,
82:        uint256,
83:        bytes calldata
84:    ) external pure returns (bytes4) {
85:        return 0x150b7a02; //IERC721Receiver.onERC721Received.selector;
86:    }
87:}
```

Description

jayphbee : VirtualWallet implements the `onERC721Received` function but not implement the `onERC1155Received` and `onERC1155BatchReceived` function to indicate it is willing to receive ERC1155 token.
0xzoobi : `VirtualWallet.sol` implements a `onERC721Received` to confirm that the contract is willing to receive ERC721 tokens.

Similarly there is a ERC1155 based tokens as well which is popular standard used generally in Metaverse based projects. Currently the `VirtualWallet.sol` will not accept such tokens.

Recommendation

jayphbee : implment the `onERC1155Received` and `onERC1155BatchReceived` function.
0xzoobi : Implement a `onERC1155Received` and `onERC1155BatchReceived`.

Example code

```
function onERC1155Received(address, address, uint256, uint256, bytes memory) public virtual returns
(bytes4) {
   return this.onERC1155Received.selector;
}

function onERC1155BatchReceived(address, address, uint256[] memory, uint256[] memory, bytes memory)
public virtual returns (bytes4) {
   return this.onERC1155BatchReceived.selector;
}
```

Reference - https://forum.moralis.io/t/how-to-implement-erc1155received-function/5492/1

Client Response

Fixed

DAP-25:Cache array length outside of loop in `PayDB` contract

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	hunya

Code Reference

- code/contracts/core/PayDB.sol#L42
- code/contracts/core/PayDB.sol#L76
- code/contracts/core/PayDB.sol#L271
- code/contracts/core/PayDB.sol#L318
- code/contracts/core/PayDB.sol#L347
- code/contracts/core/PayDB.sol#L371
- code/contracts/core/PayDB.sol#L426

```
42:         for (uint256 i = 0; i < cparam.length; i++) {

76:         for (uint256 i = 0; i < cparam.length; i++) {

271:         for (uint256 i = 0; i < eparam.length; i++) {

318:         for (uint256 i = 0; i < eparam.length; i++) {

347:         for (uint256 i = 0; i < cparam.length; i++) {

371:         for (uint256 i = 0; i < cparam.length; i++) {

426:         for (uint256 i = 0; i < eparam.length; i++) {
```

Description

hunya : There's no cache of array length of loop in function `createSrcOrder`、`createSrcOrderETH`、`executeDstOrderETH`、`executecDstOrder`、`cancelOrderETH`、`cancelOrder`、`_executeIsolateOrder`. If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

Recommendation

hunya : Cache array length outside of loop.

Consider below fix in the `PayDB.createSrcOrder()` function

```
function createSrcOrder(
    CreateOrderParam[] calldata cparam,
    uint256 code,
    address wallet,
    address service,
    bytes calldata data
) external override {
    uint256 cparamArrayLength = cparam.length;
    for (uint256 i = 0; i < cparamArrayLength; i++) {
        TransferHelper.safeTransferFrom(
            cparam[i].tokenIn,
            msg.sender,
            cparam[i].node,
            cparam[i].amountIn
        );
        if (i == cparamArrayLength - 1) {
            // The last order contains execution data.
            _createSrcOrder(cparam[i], code, wallet, service, data);
        } else {
            // Payment only.
            _createSrcOrder(
                cparam[i],
                code,
                wallet,
                address(0),
                new bytes(0)
            );
        }
    }
}
```

Client Response

Fixed

DAP-26:Events are not indexed

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	0xzoobi

Code Reference

- code/contracts/core/vwmanager/VWManagersReader.sol#L11
- code/contracts/core/vwmanager/VWManagersReader.sol#L12
- code/contracts/governance/PayLock.sol#L41-L44
- code/contracts/governance/PayLock.sol#L46-L48

```
11:     event AddVWManager(address _VWManager);

12:     event UpdateVWManager(uint256 index, address _VWManager);

41:     event TokenConfiged(
42:         address token,
43:         bool valid
44:     );

46:     event WithdrawPendingTime(
47:         uint peroid
48:     );
```

Description

0xzoobi : Indexed parameters are searchable parameters and help to query events. Non-Indexed parameters are regular parameters passed to an event that is not searchable and are only used to log the messages to the blockchain.

Each event can use three indexed fields.

Recommendation

0xzoobi : use the `indexed` keyword for the parameters you want the be searchable via indexing protocols like the Graph.

Consider below example fix

```
event Deposited(uint256 indexed epoch, uint256 indexed amount);

function deposit()  external {
        emit Deposited(epoch, amount);
```

Client Response

Fixed

DAP-27:IService does not check return value on `execute`

Category	Severity	Status	Contributor
Logical	Informational	Fixed	0xzoobi

Code Reference

- code/contracts/core/interfaces/IService.sol#L5

```
5:    function execute(uint code, bytes calldata data, address node) external;
```

Description

0xzoobi : interface IService's `execute` function call does not have a return value. I see other instances of execute checking for a Boolean return value. I don't see any security issue with respect to this but it is a good practice to always have a return value check implemented.

Recommendation

0xzoobi : Add a Boolean return value check.

Example: contracts/core/interfaces/IVirtualWallet.sol#L7-L12

Client Response

Fixed

DAP-28:Improve the error messages

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	Yaodao, 0xzoobi

Code Reference

- code/contracts/peripherals/common/ERC20Service.sol#L17
- code/contracts/core/PayDB.sol#L108
- code/contracts/governance/PayLock.sol#L115
- code/contracts/peripherals/common/HybridPayService.sol#L168
- code/contracts/core/PayDB.sol#L294
- code/contracts/core/PayDB.sol#L360
- code/contracts/core/PayDB.sol#L475

```
17:         require(block.chainid == dstChainId);

108:         require(msg.value == totalEth);

115:         require(req.status == 1 && req.submitTime + withdrawPendingTime <= block.timestamp, "CLA
IM_PENDING");

168:         require(block.chainid == dstChainId);

294:         require(msg.value == totalETH);

360:         require(msg.value == totalETH);

475:         require(msg.value == totalETH);
```

Description

Yaodao : The following checks are lack of the error message.

For example

```
    require(block.chainid == dstChainId);
```

0xzoobi : This issue tracks the improvements and other things that can be made as part of the contract.

Recommendation

Yaodao : Recommend adding the corresponding error message.

0xzoobi : Make changes to the reported issues.

1. The current revert message for require(req.status == 1 && req.submitTime + withdrawPendingTime <= block.timestamp, "CLAIM_PENDING"); is not clear. Update "CLAIM_PENDING" to "CLAIM_PERIOD_OVER".

Client Response

Fixed

DAP-29:Lack of zero address checking

Category	Severity	Status	Contributor
Logical	Informational	Fixed	danielt

Code Reference

- code/contracts/core/VirtualWallet.sol#L52-L67

```
52:     function payFee(
53:         address _feeToken,
54:         uint256 _gasFee,
55:         address _feeReceiver
56:     ) external override onlyVWManager returns (bool success) {
57:         if (_feeToken == address(0)) {
58:             //  TransferHelper.safeTransferETH(_feeReceiver, _gasFee);
59:             (success, ) = _feeReceiver.call{value: _gasFee}("");
60:         } else {
61:             //  TransferHelper.safeTransfer(_feeToken, _feeReceiver, _gasFee);
62:             (bool res, bytes memory data) = _feeToken.call(
63:                 abi.encodeWithSelector(0xa9059cbb, _feeReceiver, _gasFee)
64:             );
65:             success = res && (data.length == 0 || abi.decode(data, (bool)));
66:         }
67:     }
```

Description

danielt : In the `VirtualWallet` contract, the role `manager` will transfer tokens to the `_feeReceiver`. However, lacks zero address checking on the `_feeReceiver` account to prevent potential token loss.

Recommendation

danielt : Adding zero address checking on the `_feeReceiver` account.

Client Response

Fixed

DAP-30:Missing emit event

Category	Severity	Status	Contributor
Code Style	**Informational**	**Fixed**	**danielt, 0xgm, 0xzoobi**

Code Reference

- code/contracts/core/interfaces/IVirtualWalletV2.sol#L18
- code/contracts/core/PayDB.sol#L29-L31
- code/contracts/core/VirtualWallet.sol#L66
- code/contracts/peripherals/reward/RewardPool.sol#L112-L117
- code/contracts/peripherals/reward/RewardPool.sol#L119-L124
- code/contracts/core/vwmanager/VWManager.sol#L141-L149

```
18:}

29:    function setDefaultVWManager(address _manager) external onlyOwner {
30:        VWManager = _manager;
31:    }

66:        }

112:    function setRewardConfig(RewardConfig calldata _rewardConfig)
113:        external
114:        onlyOwner
115:    {
116:        rewardConfig = _rewardConfig;
117:    }

119:    function setClaimConfig(address token, ClaimConfig calldata claimConfig)
120:        external
121:        onlyOwner
122:    {
123:        claimConfigs[token] = claimConfig;
124:    }

141:    function configFee(
142:        bool _protocolFeeOpened,
143:        address _feeVault,
144:        uint256 _feeProportion
145:    ) external onlyOwner {
146:        protocolFeeOpened = _protocolFeeOpened;
147:        feeVault = _feeVault;
148:        feeProportion = _feeProportion;
149:    }
```

Description

danielt : In the `PayDB` contract, the below key function lack of missing event:

- `setDefaultVWManager`

In the `RewardPool` contract, the below key function lack of missing event:

- `setRewardConfig`
- `setClaimConfig`

Emitting an event when updating the state variable is important to track the state of the contract.
0xgm : Consider emitting an event for the `VirtualWallet.payFee()` function. This will better indicate that a fee has

been paid with the correct parameters. Due that fees can be paid in an array of tokens, this becomes even more important for tracking during transaction executions in a cross-chain application.

Since there are several instances of calling a function that pays a fee, i.e. `_walletPayFee()` and `sendAndPayFee()`, this event should be emitted from the underlying `VirtualWallet.payFee()` function, so that it may be written once and emitted during any implementation of it.

0xzoobi : Every project must follow the template wherein they emit events on important changes and updates happening in the dapp. Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes. For Example : The Graph

Recommendation

danielt : Emit events for key functions that update the state variables.

0xgm : The new event should be added to the `IVirtualWalletV2` interface that the `VirtuatlWallet` contract inherits so that it can then be emitted during the `payFee` function.

```
interface IVirtualWalletV2 {
   // ...
   event FeePaid(
      address feeToken,
      uint256 gasFee,
      address feeReceiver
   );
}
```

The event should be emitted if the calls were successful.

```
contract VirtualWallet {
    function payFee(
        address _feeToken,
        uint256 _gasFee,
        address _feeReceiver
    ) external override onlyVWManager returns (bool success) {
        if (_feeToken == address(0)) {
            //  TransferHelper.safeTransferETH(_feeReceiver, _gasFee);
            (success, ) = _feeReceiver.call{value: _gasFee}("");
        } else {
            //  TransferHelper.safeTransfer(_feeToken, _feeReceiver, _gasFee);
            (bool res, bytes memory data) = _feeToken.call(
                abi.encodeWithSelector(0xa9059cbb, _feeReceiver, _gasFee)
            );
            success = res && (data.length == 0 || abi.decode(data, (bool)));
        }

       // add this
       if (success) {
         emit FeePaid(_feeToken, _gasFee, _feeReceiver);
       }
       return success;
    }
}
```

0xzoobi : Add an event when the function executes.

Client Response

Fixed.reward contract is designed to be called through vwmanager

DAP-31:Remove unnecessary receive or fallback function

Category	Severity	Status	Contributor
Logical	Informational	Declined	jayphbee

Code Reference

- code/contracts/core/VirtualWallet.sol#L72
- code/contracts/peripherals/reward/RewardPool.sol#L128
- code/contracts/core/PayDB.sol#L496

```
72:    fallback() external payable {}

128:    fallback() external payable {}

496:    fallback() external payable {}
```

Description

jayphbee : * `PayDB` contract is not intended to receive ether, so the `receive` and `fallback` function should be removed.

- `RewardPool` contract implements `receive` and `fallback`, just keep the `receive` is enough to receive ether.
- `VirtualWallet` contract is same as `RewardPool`.

Recommendation

jayphbee : Remove unnecessary `receive` or `fallback` function.

Client Response

Declined.paydb will receive eth to in method createSrcOrderETH()

DAP-32:Repeated functions in different contracts.

Category	Severity	Status	Contributor
Gas Optimization	Informational	Acknowledged	danielt

Code Reference

- code/contracts/libraries/OrderId.sol#L7-L17
- code/contracts/libraries/VWCode.sol#L19-L55

```
7:     function genCode(
8:         uint128 nonce, uint32 time, uint32 srcChainId, uint32 dstChainId, uint16 oType, uint16 fla
g
9:     ) internal pure returns (uint code){
10:         code = (uint(nonce) << 128) + (uint(time) << 96) + (uint(srcChainId) << 64) + (uint(dstCh
ainId) << 32) + (uint(oType) << 16) + uint(flag);
11:     }
12:
13:     function chainidsAndExpTime(uint code) internal pure returns (uint dstChainId, uint srcChainI
d, uint time){
14:         dstChainId = (code >> 32) & ((1 << 32) - 1);
15:         srcChainId = (code >> 64) & ((1 << 32) - 1);
16:         time = (code >> 96) & ((1 << 32) - 1);
17:     }

19:     function genCode(
20:         uint128 nonce,
21:         uint32 time,
22:         uint32 srcChainId,
23:         uint32 dstChainId,
24:         uint16 action,
25:         uint16 flag
26:     ) internal pure returns (uint256 code) {
27:         code =
28:             (uint256(nonce) << 128) +
29:             (uint256(time) << 96) +
30:             (uint256(srcChainId) << 64) +
31:             (uint256(dstChainId) << 32) +
32:             (uint256(action) << 16) +
33:             uint256(flag);
34:     }
35:
36:     /**
37:      *@dev Extracts dstChainId, srcChainId, and time values from a given code parameter
38:      *@param code uint value
39:      *@return dstChainId The destination chain ID
40:      *@return srcChainId The source chain ID
41:      *@return time The expiration time
42:      */
43:     function chainidsAndExpTime(uint256 code)
44:         internal
```

```
45:         pure
46:         returns (
47:             uint256 dstChainId,
48:             uint256 srcChainId,
49:             uint256 time
50:         )
51:     {
52:         dstChainId = (code >> 32) & ((1 << 32) - 1);
53:         srcChainId = (code >> 64) & ((1 << 32) - 1);
54:         time = (code >> 96) & ((1 << 32) - 1);
55:     }
```

Description

danielt : Both the `VWCode` contract and the `OrderId` contract have the same functions: `genCode` and `chainidsAndExpTime`.

Reducing redundant functions is good for gas saving and readability.

Recommendation

danielt : Reducing redundant functions.

Client Response

Acknowledged.Does not matter,maybe payorderid will change encode method in the future

DAP-33:Unused import

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	Yaodao

Code Reference

- code/contracts/governance/PayLock.sol#L4
- code/contracts/peripherals/common/HybridPayService.sol#L6
- code/contracts/peripherals/common/HybridPayService.sol#L7
- code/contracts/governance/PayLock.sol#L8

```
4:import "../core/PayDB.sol";

6:import "../../libraries/HeaderLibrary.sol";

7:import "../../libraries/TransferHelper.sol";

8:import "@openzeppelin/contracts/utils/cryptography/ECDSA.sol";
```

Description

Yaodao : These files are imported but not used.

Recommendation

Yaodao : Recommend removing unused import.

Client Response

Fixed

DAP-34:Unused return value

Category	Severity	Status	Contributor
Logical	Informational	Fixed	danielt

Code Reference

- code/contracts/core/vwmanager/VWManager.sol#L200-L213

```
200:         if (protocolFeeOpened) {
201:             splitAndSendFee(
202:                 eParam.wallet,
203:                 eParam.payToken,
204:                 gasFee,
205:                 feeReceiver
206:             );
207:         } else {
208:             res = IVirtualWalletV2(eParam.wallet).payFee(
209:                 eParam.payToken,
210:                 gasFee,
211:                 feeReceiver
212:             );
213:         }
```

Description

danielt : In the `VWManager.sol` contract, the `execute` function will call the `splitAndSendFee` function or the `payFee` function to pay the fee, but only check the return value of the `payFee` function, the return value of the `splitAndSendFee` function is ignored:

```
        if (protocolFeeOpened) {
            splitAndSendFee(
                eParam.wallet,
                eParam.payToken,
                gasFee,
                feeReceiver
            );
        } else {
            res = IVirtualWalletV2(eParam.wallet).payFee(
                eParam.payToken,
                gasFee,
                feeReceiver
            );
        }
```

On the other hand, the internal function `splitAndSendFee` is only invoked in the `execute` function, if the return value of the `splitAndSendFee` is not used by the `execute` function and useless, it is recommended to refactor the `splitAndSendFee` function and not return a value.

Recommendation

danielt : Recommend also checking the return value of the `splitAndSendFee` function.

Client Response

Fixed

DAP-35: `feeProportion` should be bounded.

Category	Severity	Status	Contributor
Logical	Informational	Fixed	jayphbee

Code Reference

- code/contracts/core/vwmanager/VWManager.sol#L148

```
148:         feeProportion = _feeProportion;
```

Description

jayphbee : There should be an upper bound for `feeProportion` in the `configFee` function. If `feeProportion` accidently set an unreasonable value, user will pay unreasonable fee.

```
    function configFee(
        bool _protocolFeeOpened,
        address _feeVault,
        uint256 _feeProportion
    ) external onlyOwner {
        protocolFeeOpened = _protocolFeeOpened;
        feeVault = _feeVault;
        feeProportion = _feeProportion;
    }
```

Recommendation

jayphbee : Add an upper bound for the `feeProportion`.

Client Response

Fixed

DAP-36:redundant use of `receive` and `fallback` in the same contract

Category	Severity	Status	Contributor
Gas Optimization	Informational	Acknowledged	0xzoobi

Code Reference

- code/contracts/core/VirtualWallet.sol#L70
- code/contracts/core/PayDB.sol#L70

```
70:         address service,

70:     receive() external payable {}
```

Description

0xzoobi : The `receive` is used when a contract wants to receive ether and `fallback` is used for the same purpose but it also accepts `calldata`.

Using both of them may be required for a condition shown below wherein fallback invokes function `doTask1()` and receive invokes `doTask2()` but in the current scenario using `fallback` is sufficient.

```
fallback() external payable {
    result = doTask1(msg.data);
}

receive() external payable {
    doTask2();
}
```

Recommendation

0xzoobi : Remove the `receive()` function.

Client Response

Acknowledged

DAP-37:use `external` identifier for functions instead of `public`

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	Yaodao, 0xzoobi

Code Reference

- code/contracts/core/PayDB.sol#L251
- code/contracts/core/PayDB.sol#L251-L255
- code/contracts/core/PayDB.sol#L299
- code/contracts/core/PayDB.sol#L299-L303

```
251:    function executeDstOrderETH(

251:    function executeDstOrderETH(
252:        ExeOrderParam[] calldata eparam,
253:        bytes calldata data,
254:        bytes calldata serviceSignature
255:    ) public payable override {

299:    function executeDstOrder(

299:    function executeDstOrder(
300:        ExeOrderParam[] calldata eparam,
301:        bytes calldata data,
302:        bytes calldata serviceSignature
303:    ) public override {
```

Description

Yaodao : The following functions are declared as `public`, contain array function arguments, and are not invoked in any of the contracts contained within the project's scope. Functions that are never called internally within the contract should have `external` visibility.
0xzoobi : functions having `public` identifier can be called via EOA/contracts and also can be called within the same contract as well. functions having `external` identifier can be only called via EOA/contracts and not within the same contract.

Recommendation

Yaodao : Recommend setting visibility specifiers to `external` to optimize the gas cost of the function.
0xzoobi : The functions for which there is no need to called by the contract itself can be marked `external`. This can help in optimizing the contract overall. Apart from the below suggestions, please review all the functions and make the changes if required.

Client Response

Fixed

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.