Competitive Security Assessment

dappOSP4

Aug 17th, 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	dappOSP4
Platform & Language	Solidity
Codebase	• https://github.com/DappOSDao/contracts-core • audit commit - bbd6741d0d48a64a7947d4417b03af3ed73323f0 • final commit - bd0eadeddcba3b86c469c35df131819da0a71ba8
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Code Vulnerability Review Summary

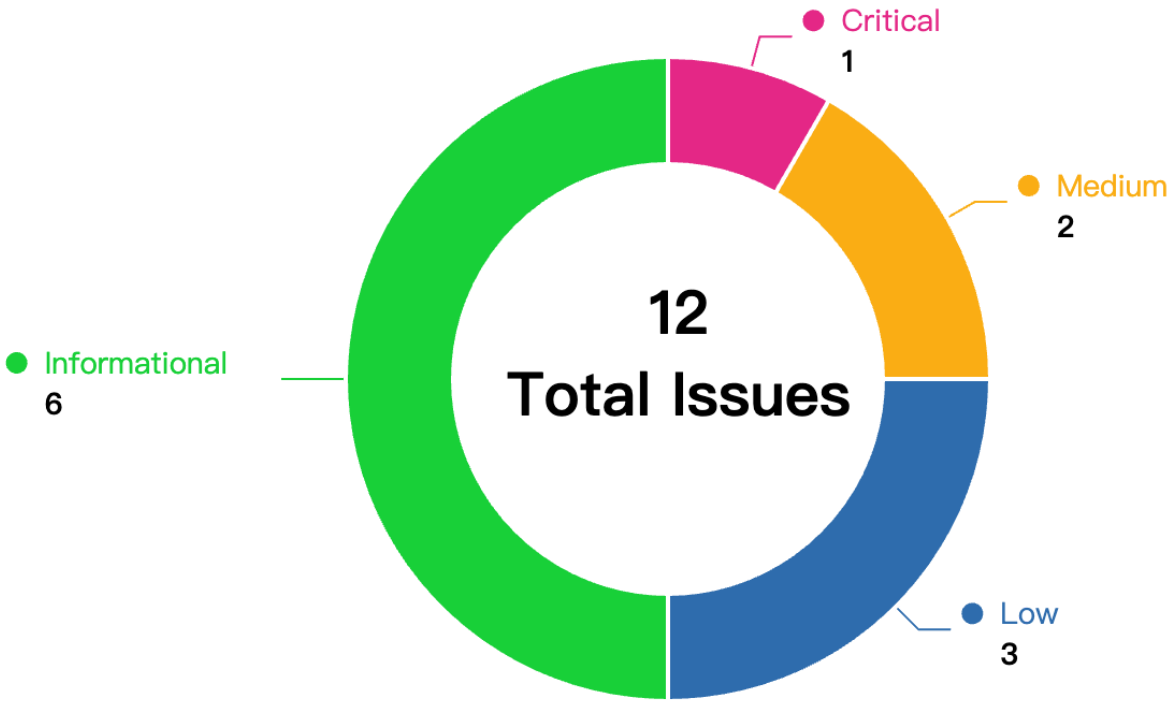
Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	1	0	0	1	0	0
Medium	2	0	0	2	0	0
Low	3	0	2	0	0	1
Informational	6	0	0	6	0	0

Audit Scope

File	SHA256 Hash
contracts/core/PayDB.sol	bbc2b675472af398ac91fe90fecc578f72c3f5724c7887ad1541db2b8afed074
contracts/core/VirtualWallet.sol	f754eb5ecd5e7d943cbe492dbdd59b9b1e9400f71278bc500028f97b32e46cd7
contracts/core/interfaces/IPayDB.sol	0637984fc88a728f9c23bae8fffdc1f37583264763c085e90c7064f36fe01268
contracts/core/interfaces/IService.sol	f79584d8869a04cc92d23b4c9b1c7993cb4cd041d8ddccc6acab647cc6bcbd15
contracts/core/interfaces/IVWManager.sol	c5a1390935fbb213090816b333ced8c5c33cdd62d3186c6769cf65b213b7e4c9
contracts/core/interfaces/IVWManagerStorage.sol	431b570dd0851fd6820e0a39dc767d90dc63cb8d0f66adfc04c1b5b9982edf5c
contracts/core/interfaces/IVWResetter.sol	de8b732bd43bea7a985a95aed575319ffb109b8adca7a555860219831c46cbfa
contracts/core/interfaces/IVirtualWalletV2.sol	0736ec68e6124da1f5a671d0b91efc1c4a12e610ab3fbcacb26883ea4857cb28
contracts/core/vwmanager/VWManager.sol	26670c6ecd369f878f2590f53fb16ab36928614360aaaa2e6700738d57652224
contracts/core/vwmanager/VWManagerService.sol	092184ae44f78cf55bc901df920188a24e883c6d4a5c0bc0a1178be117c6183e
contracts/core/vwmanager/WalletDeployer.sol	ca3f0bf1724f22d570dee83791ba9271c4ae104a5e312b37827dac1e504ad8d5
contracts/core/vwmanager/storage/VWManagerStorage.sol	2b21671f403c8440cace423f72514eee6cc19691ef3662d4a291885ce10ebd8e
contracts/governance/PayLock.sol	6d2ffa50fc82cf00a013b7f5340ff2465df983c42985d539fa2b60fcaf473c62
contracts/libraries/LowGasSafeMath.sol	76d29d0cb3c13f6c52d73bae5c16ff3ef035764e8275948fda551c00be351ec0
contracts/libraries/OrderId.sol	5729d3e0bf27cfb07b71f4e539139468efd92ab936d6113da38be7950b16000d

contracts/libraries/SafeCast.sol	**ca3e483bd0394b8f8457f7088d3058f26571057081abac75c599bcd03a69c895**
contracts/libraries/SignLibrary.sol	**1653c46efe86484b8121e51d3861bd5f0e2f2a48e9e61bdd104282c403f9e7e2**
contracts/libraries/TransferHelper.sol	**4aa11019758c68e6ff327bed3b4c114d3ebb9a5eb64e2f25215152ff6203df86**
contracts/libraries/VWCode.sol	**5afa464d3eed682671784988afaa6f585b66ddfbe1dba4f73c8d3f19ac846b9a**

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
DAP-1	Order ExpiryTime should be less than node withdrawal time `PayDB::creatSrcOrder()` function	Race Condition	Critical	Fixed	0xffchain
DAP-2	Node evades punishment when valid token disabled `PayLock::punish()` function	Logical	Medium	Fixed	0xffchain
DAP-3	Node Block Stuff Chain to steal user funds `PayLock::claim()` function	Race Condition	Medium	Fixed	0xffchain

DAP-4	Return value not used	Logical	Low	Acknowledged	danielt
DAP-5	Lack of input validation	Logical	Low	Declined	Hacker007
DAP-6	Unnecessary storage consumption due to claimed withdrawRequests remaining in the contract's state.	Gas Optimization	Low	Acknowledged	crjr0629
DAP-7	Return inaccurate error message	Logical	Informational	Fixed	Yaodao, Hacker007
DAP-8	Remove unused imports	Code Style	Informational	Fixed	Yaodao
DAP-9	Gas optimization: simplified expression	Gas Optimization	Informational	Fixed	danielt
DAP-10	Unlocked Pragma Version	Code Style	Informational	Fixed	Yaodao
DAP-11	Gas Optimization: Revert early in `verifyProof()`	Gas Optimization	Informational	Fixed	Hacker007
DAP-12	Remove unused function in tranferHelper	Code Style	Informational	Fixed	crjr0629

DAP-1:Order ExpiryTime should be less than node withdrawal time `PayDB::createSrcOrder()` function

Category	Severity	Client Response	Contributor
Race Condition	Critical	Fixed	0xffchain

Code Reference

- code/contracts/core/PayDB.sol#L415

```
415:(, uint256 srcChainId, uint256 time) = OrderId.chainidsAndExpTime(
```

Description

0xffchain : Order expiry time as included in each cross-chain transfer in PayDB functions that perform interchain transfers should be validated to be less than the node request withdrawal pending time. Because if withdrawal request pending time is less than order expiry time, it means that a user can make a request and a node place withdrawal after seeing the request and since the pending time is less than order execution time, node can claim its security stake while still having assets deposited by user into its node.

Recommendation

0xffchain : Should have a validation that checks time(execution expiry time) is less than `withdrawPendingTime` time.

Client Response

Fixed.

DAP-2:Node evades punishment when valid token disabled `PayLock::punish()` function

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	0xffchain

Code Reference

- code/contracts/governance/PayLock.sol#L141

```
141:require(validTokens[tokens[i]], "INVALID_TOKEN");
```

Description

0xffchain : The punish function allows a node to evade punishment after a previously valid token becomes invalid while still having orders assigned to the node.

`require(validTokens[tokens[i]], "INVALID_TOKEN");`

The function checks only for valid token.

This is in stark contrast with `submitWithdrawRequest` function and `claim` function in PayLock that allows the node to still withdraw and request withdrawal when a previously valid token becomes invalid.

So if a Node should be able to gets its token balance back after the previously valid token now invalidated by the contract, the user should be able to do same as well.

Recommendation

0xffchain : The Validation should be removed this example lines added:

```
if(bal.numOnWithdraw != 0 && bal.numTotal != 0 ){// code here}
```

Client Response

Fixed.

DAP-3:Node Block Stuff Chain to steal user funds `PayLock::claim()` function

Category	Severity	Client Response	Contributor
Race Condition	Medium	Fixed	0xffchain

Code Reference

- code/contracts/governance/PayLock.sol#L115
- code/contracts/core/PayDB.sol#L384

```
115:function claim(uint requestID) external {

384:function _createSrcOrder(
```

Description

0xffchain : A node can block stuff the difference in time between order expiry time and `withdrawPendingTime` , in other to avoid both being punished for not executing an order and also absconding with user funds and locked deposits in PayLock.

Recommendation

0xffchain : Make sure The window between `submitWithdrawRequest` and order expiration time is wide enough that it makes block stuffing unprofitable.

Client Response

Fixed.

DAP-4:Return value not used

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	danielt

Code Reference

- code/contracts/core/PayDB.sol#L146
- code/contracts/core/PayDB.sol#L379

```
146:IVWManager(vwExeParam.manager).execute(

379:IVWManager(vwExeParam.manager).execute(wallet, vwExeParam);
```

Description

danielt : The `executeDstOrderETH` function and the `_executeIsolateOrder` function invoke the `execute` function, which has a return value `resCode`:

```
    //IVWManager.sol
    function execute(
        address wallet,
        VWExecuteParam calldata eParam
    ) external returns (uint resCode);

    function _executeIsolateOrder(
        address receiver,
        address wallet,
        ExePayOrderParam[] calldata eparams,
        IVWManager.VWExecuteParam calldata vwExeParam
    ) internal {
        ...
        if (vwExeParam.code != 0 && vwExeParam.serviceSignature.length > 0) {
            IVWManager(vwExeParam.manager).execute(wallet, vwExeParam);
        }
        ...
    }
```

The return value of the `execute` function is recommended to be checked to prevent any potential failure or exception of executing the `execute` function, like what did in the `tryExecuteDstOrderETH` function:

```
    function tryExecuteDstOrderETH(
        address orderOwner,
        address receiver,
        address wallet,
        ExePayOrderParam[] calldata eparams,
        IVWManager.VWExecuteParam calldata vwExeParam
    ) external payable override {
        ...
            try IVWManager(vwExeParam.manager).execute(
                wallet,
                vwExeParam
            ) returns(uint) {
            } catch Error(string memory failReason){
                emit VWFailedReason(failReason);
            }
        ...
    }
```

Recommendation

danielt : Consider validating the return value of the `execute` function.

Client Response

Acknowledged.We provide two different sets of VW execution entry points for different execution purposes. In the executeDstOrderETH method, it is necessary to ensure the successful execution of VWManager, so there is no need to add a return value in the method.

DAP-5:Lack of input validation

Category	Severity	Client Response	Contributor
Logical	Low	Declined	Hacker007

Code Reference

- code/contracts/core/vwmanager/VWManager.sol#L135

```
135:function requestConfigSrcChain(uint256 _chainId, bool _support, address _verifyingContract) exte
rnal onlyOwner {
```

Description

Hacker007 : The function `requestConfigSrcChain` from contract `VWManager` does not check if the input address `_verifyingContract` is a contract address, which may bring unexpected errors.

Recommendation

Hacker007 : Check if the input parameter `_verifyingContract` is a contract address.

Client Response

Declined.In the system design of dappOS, the _verifyingContract parameter is passed with the address of VWManager on another chain, so it cannot perform a contract existence check.

DAP-6:Unnecessary storage consumption due to claimed withdrawRequests remaining in the contract's state.

Category	Severity	Client Response	Contributor
Gas Optimization	Low	Acknowledged	crjr0629

Code Reference

- code/contracts/governance/PayLock.sol#L32-L33

```
32:mapping(uint => WithdrawRequest) public withdrawRequests;
```

Description

crjr0629 : Completed withdrawRequests remain permanently in the `withdrawRequests` mapping, leading to inefficiencies and potential state bloat. (There is no way to remove a withdrawRequest from withdrawRequests mapping which can lead to a problem). Over time, as more requests are completed, this could contribute to a continuous growth in the contract's storage size, leading to increased costs and potential state bloat. While not a direct security vulnerability, this behavior could have implications for the efficiency, usability, and privacy of the contract.

Recommendation

crjr0629 : Completed requests from the withdrawRequests mapping can be deleted in the claim() function, because if one request is "claimed", you wont be able to claim anymore. Also, the public variable `numWithdrawRequest` should be decremented.

Client Response

Acknowledged.We intend to keep a record of all requests.

DAP-7:Return inaccurate error message

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	Yaodao, Hacker007

Code Reference

- code/contracts/core/vwmanager/VWManager.sol#L82
- code/contracts/core/vwmanager/VWManager.sol#L122

```
82:require(dstChainId == block.chainid, 'E8');

122:require(_chainId != block.chainid, 'E21');
```

Description

Yaodao : In the function `configSrcChain()`, the error message `E21` is used for the check `_chainId != block.chainid` which checks the chainid. However, according to the doc `README.md`, the error message `E21` is used for `feePortion` limit instead of chainid check.

```
E21: feePortion is limited in 2000
```

Hacker007 : Per the document, the error code `E3` represents `Not the dst chain`, However, in the function `verifyProof()`, if `dstChainId` isn't equal to `block.chainid`, an incorrect error code `E8` will be returned, which represents `Payment src chain error or expired`.

```
require(dstChainId == block.chainid, 'E8');
```

Recommendation

Yaodao : Recommend using the corresponding error message.
Hacker007 : throw the error code `E3` instead of `E8`.

Client Response

Fixed.change error code

DAP-8:Remove unused imports

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	Yaodao

Code Reference

- code/contracts/libraries/SignLibrary.sol#L5
- code/contracts/core/VirtualWallet.sol#L10
- code/contracts/core/vwmanager/VWManager.sol#L5

```
5:import "../core/interfaces/IWalletOwner.sol";

5:import '../interfaces/IService.sol';

10:import "../libraries/TransferHelper.sol";
```

Description

Yaodao : The contract `SignLibrary` includes the following unnecessary imports:

```
import "../core/interfaces/IWalletOwner.sol";
```

The contract `VirtualWallet` includes the following unnecessary imports:

```
import "../libraries/TransferHelper.sol";
```

The contract `VWManager` includes the following unnecessary imports:

```
import '../interfaces/IService.sol';
```

Recommendation

Yaodao : Recommend removing the import statement to save on deployment gas costs.

Client Response

Fixed.removed ununsed imports

DAP-9:Gas optimization: simplified expression

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	danielt

Code Reference

- code/contracts/core/PayDB.sol#L436-L438

```
436:cparams[i].tokenOut,
437:                        receiver,
438:                        workFlowHash
```

Description

danielt : In the `_createSrcOrder` function of the `PayDB` contract, a memory variable `_order` is declared with the value of the storage variable `order`, however, the memory variable `_order` is only used once.

```
    function _createSrcOrder(
        address _orderOwner,
        address wallet,
        address receiver,
        CreatePayOrderParam[] calldata cparams,
        VwOrderDetail calldata vwDetail,
        CallParam calldata callParam
    ) internal {
        ...
                require(uint256(_order.status) == 0, "E7");
                (order.node, order.status, order.orderDataHash) = (
                    cparams[i].node,
                    STATUS_RECEIVED,
                    orderDetail
                );
            }
            emit OrderCreated(
                _orderOwner,
                cparams[i].node,
                cparams[i].payOrderId,
                msg.sender,
                receiver,
                cparams[i].amountIn,
                cparams[i].tokenIn,
                cparams[i].amountOut,
                cparams[i].tokenOut,
                vwDetail.code,
                workFlowHash
            );
        }

        require(msg.value == totalEth,"E18");

        if (callParam.nodeCallData.length > 0) {
            INodeCall(callParam.node).CreateSrcOrderCall(callParam.nodeCallData,_orderOwner,receive
r,wallet,cparams,vwDetail);
        }
    }
```

Thus, we can save gas by not declaring the memory variable `_order` and directly using the storage variable `order` in the `require` statement.

Recommendation

danielt : To save gas, we can not declare the memory variable `_order` and directly use the storage variable `order` in the `require` statement:

```
                require(uint256(order.status) == 0, "E7");
```

Client Response

Fixed. gas usage fixed

DAP-10:Unlocked Pragma Version

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	Yaodao

Code Reference

- code/contracts/core/vwmanager/WalletDeployer.sol#L3

```
3:pragma solidity ^0.8.19;
```

Description

Yaodao : Solidity files in packages have a pragma version `^0.8.19`. The caret `(^)` points to unlocked pragma, meaning the compiler will use the specified version or above.

Recommendation

Yaodao : It's good practice to use specific solidity versions to know compiler bug fixes and optimisations were enabled at the time of compiling the contracts.

Client Response

Fixed. locked solidity version

DAP-11:Gas Optimization: Revert early in `verifyProof()`

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	Hacker007

Code Reference

- code/contracts/core/vwmanager/VWManager.sol#L76-L108

```
76:*/
77:    function verifyProof(uint resCode, address wallet, VWExecuteParam calldata vweParam) internal
{
78:        address vwOwner = walletOwner[wallet];
79:        result[vwOwner][vweParam.code] = resCode;
80:
81:        (uint256 dstChainId, uint256 srcChain, uint256 expTime) = VWCode.chainidsAndExpTime(vwePa
ram.code);
82:        require(dstChainId == block.chainid, 'E8');
83:        require(block.timestamp <= expTime, 'E6');
84:        bytes32 rootHash = keccak256(
85:            abi.encode(
86:                APPROVE_SERVICE_TX_TYPEHASH,
87:                vweParam.code,
88:                keccak256(vweParam.data),
89:                vweParam.service,
90:                vweParam.gasToken,
91:                vweParam.gasTokenPrice,
92:                vweParam.priorityFee,
93:                vweParam.gasLimit,
94:                vweParam.isGateway
95:            )
96:        );
97:        if (vweParam.proof.length > 0) {
98:            rootHash = MerkleProof.processProof(vweParam.proof, rootHash);
99:            rootHash = keccak256(abi.encode(APPROVE_SERVICE_PROOF_TX_TYPEHASH, rootHash));
100:        }
101:        // srcChain is the chain where user sign the rootHash
102:        if (Address.isContract(vwOwner)) {
103:            require(IWalletOwner(vwOwner).verifyVWParam(rootHash, domainSeparator[srcChain], vwe
Param), 'E1');
104:        } else {
105:            SignLibrary.verify(vwOwner, domainSeparator[srcChain], rootHash, vweParam.serviceSig
nature);
106:        }
107:        emit TxExecuted(wallet, vwOwner, vweParam.code, rootHash, resCode);
108:    }
```

Description

Hacker007 : In the contract VWManager, if the owner disables a source chain, the corresponding `domainSeparator[srcChain]` will be set to zero. Thus, the function `verifyProof()` can return directly without calculating the MerkleProof and verifying the signature to save gas.

Recommendation

Hacker007 : check if `domainSeparator[srcChain]` is `byte(0)` before calculating roothash.

```
require(block.timestamp <= expTime, 'E6');
require(domainSeparator[srcChain] != bytes32(0), 'E8');
```

Client Response

Fixed.reverted early

DAP-12:Remove unused function in tranferHelper

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	crjr0629

Code Reference

- code/contracts/libraries/TransferHelper.sol#L33-L34
- code/contracts/libraries/TransferHelper.sol#L18-L19

```
18:function safeTransferFrom2(
19:         address token,

33:function safeTransferFrom3(
34:         address token,
```

Description

crjr0629 : functions inside safeTransferFrom2() and safeTransferFrom3() from library TransferHelper are not used, even though a logic is found

```
function deposit(address token, uint amount, address node) external {
        uint256 beforeTransfer = IERC20(token).balanceOf(address(this));
        TransferHelper.safeTransferFrom(token, msg.sender, address(this), amount);
        uint256 afterTransfer = IERC20(token).balanceOf(address(this));
        _deposit(token,  afterTransfer - beforeTransfer, node);
    }
```

where safeTransferFrom3() could have been used.

Recommendation

crjr0629 : consider deleting `safeTransferFrom3()` and `safeTransferFrom2` or using them where they could be used.

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.