Competitive Security Assessment

DappOS Phase1

Nov 9th, 2022

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 Phase1
Platform & Language	Solidity
Codebase	• https://github.com/DappOSDao/contracts-v1 • audit commit - 096edc19775ec66a5a6d836ef8351cf38cd1de7a • final commit - cda002e6787f8be7af6e507c267f2e81bee904c6
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

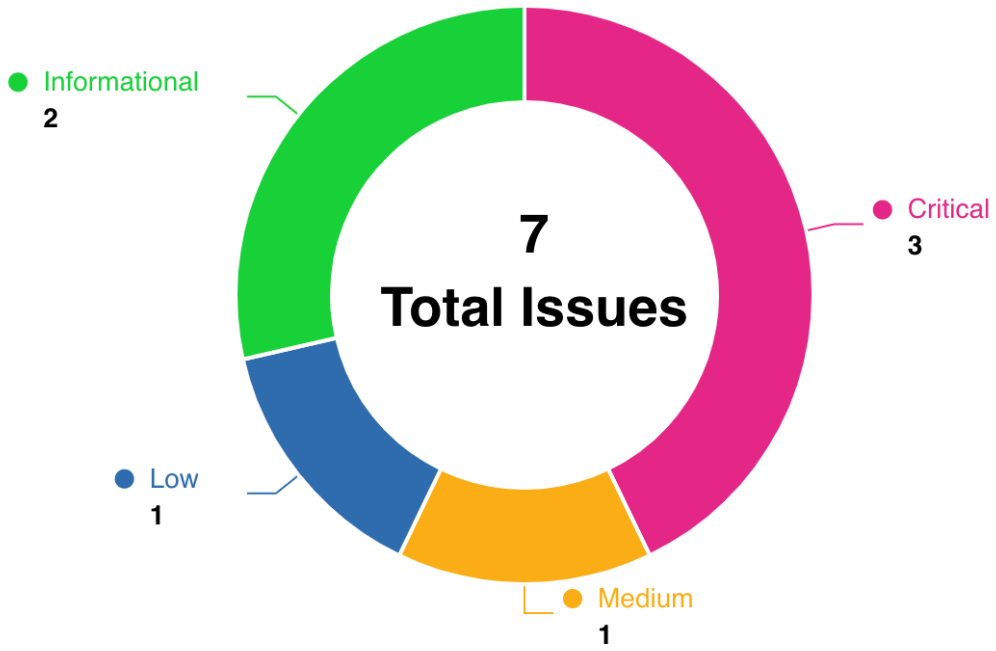
Code Vulnerability Review Summary

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

Audit Scope

File	Commit Hash
contracts/vwallet/VWManagerService.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/vwallet/VWManager.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/vwallet/libraries/SignLibrary.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/vwallet/libraries/VWCode.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/vwallet/storage/VWManagerStorage.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/vwallet/VirtualWallet.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/vwallet/WalletDeployer.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/vwallet/interfaces/IService.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/vwallet/interfaces/IVWManager.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/vwallet/interfaces/IVWManagerStorage.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/vwallet/interfaces/IVWResetter.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/vwallet/interfaces/IVirtualWallet.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/vwallet/interfaces/IWalletDeployer.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/pay/libraries/OrderId.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/pay/PayDB.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/pay/AchImpl/BalancerV1.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/pay/AchImpl/AchAuth.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/pay/AchImpl/deploy/BalancerBSC.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/pay/AchImpl/AchNode.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/pay/AchImpl/interfaces/IEncV2Router02.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/pay/AchImpl/interfaces/IEncV2Router01.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/pay/AchImpl/interfaces/IAlchemyPay02.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/pay/AchImpl/interfaces/IBalancer.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a
contracts/pay/interfaces/IPayDB.sol	096edc19775ec66a5a6d836ef8351cf38cd1de7a

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
DAP-1	`transfer` and `approve` will revert when the token doesn't follow standard implementation.	Logical	Medium	Fixed	calldata
DAP-2	Order status not checked when calling `cancelOrder` thus the already cancelled order can be cancelled again which leads to payment node lose funds.	Logical	Critical	Fixed	calldata
DAP-3	`OrderCancelled` event not emitted.	Code Style	Informational	Fixed	calldata
DAP-4	Missing immutable declaration	Logical	Informational	Fixed	xiongmao133
DAP-5	`PayDB::executeDstOrderETH` and `payDB::executeDstOrder` lack permission check	Logical	Critical	Acknowledged	Secure3
DAP-6	`VWManagerService::_resetOwner` needs to check if new owner is `address(0)`	Logical	Low	Fixed	Secure3
DAP-7	`code` should be bound to the	Signature	Critical	Fixed	Secure3

	`signer` (wallet owner) instead of the wallet to prevent signature replay attack	**Forgery or Replay**			

DAP-1: `transfer` and `approve` will revert when the token doesn't follow standard implementation.

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	code/pay/AchImpl/BalancerV1.sol#L84 code/pay/AchImpl/BalancerV1.sol#L92 code/pay/AchImpl/BalancerV1.sol#L97 code/pay/AchImpl/BalancerV1.sol#L177 code/pay/AchImpl/BalancerV1.sol#L183	Fixed	calldata

Code

```
84:                 IERC20(tokenIn).approve(anySwapRouter, type(uint).max);

92:             IERC20(tokenIn).transfer(addressBook[p1], amount);

97:                 IERC20(tokenIn).approve(cBridge, type(uint).max);

177:             IERC20(tokenIn).transfer(msg.sender, amount);

183:              IERC20(token).approve(spender, type(uint).max);
```

Description

calldata : Tokens like USDT on ethereum mainnet is not compitible with the standard ERC20. Its `approve` and `transfer` methods have no return value, thus if using the standard ERC20 interface will lead to revert. There are several places in the `BalancerV1` contract use the `approve` and `transfer` methods.

Recommendation

calldata : Use openzeppelin SafeERC20 library's `safeApprove` and `safeTransfer` instead of the standard implementation.

Client Response

Fixed. Used `SafeERC20.safeTransfer()` for the functions.

DAP-2:Order status not checked when calling `cancelOrder` thus the already cancelled order can be cancelled again which leads to payment node lose funds.

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	code/contracts/pay/PayDB.sol#L224	Fixed	calldata

Code

```
224:         require(_order.node == msg.sender && _order.orderDataHash == orderDetail, "E9");
```

Description

calldata : When perform `cancelOrder`, funds are first tranferred to the `from` address and then follow the `_cancelOrder` method, the order status not checked in `_cancelOrder`, however.

```
require(_order.node == msg.sender && _order.orderDataHash == orderDetail, "E9")
```

This line of code only check the `node` and `orderDataHash`, ignoring the `status` check. So the payment node will lose funds if the same parameter used to call `cancelOrder` more than once. The `from` address will receive more than it should.

Recommendation

calldata : check the order status when cancel order.

```
require(_order.node == msg.sender && _order.orderDataHash == orderDetail && _order.status != STATUS_CANCELLED, "E9")
```

Client Response

Fixed. Added `_order.status != STATUS_CANCELLED` in the `require` statement to make sure order cannot be cancelled again.

DAP-3: `OrderCancelled` event not emitted.

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	code/contracts/pay/PayDB.sol#L226	Fixed	calldata

Code

```
225:         order.status = STATUS_CANCELLED;
```

Description

calldata : The `OrderCancelled` event is defined in `IPayDB.sol` but never used. It should be emitted when the order is cancelled. The `OrderCancelled` event should be an essential offchain data source to account for the payment node behavior.

Recommendation

calldata : emit the `OrderCancelled` event when `cancelOrder` executed.

Client Response

Fixed. Added `emit OrderCancelled(_order.node, cparam.payOrderId);` in the `_cancelOrder()` function.

DAP-4:Missing immutable declaration

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	code/contracts/pay/PayDB.sol#L22	Fixed	xiongmao133

Code

```
22:     address public VWManager;
```

Description

xiongmao133 : Confirm that the VWManager address does not change. It should be immutable

Recommendation

xiongmao133 : Add immutable decoration

Client Response

Fixed. Declared as `address public immutable VWManager;`.

DAP-5: `PayDB::executeDstOrderETH` and `payDB::executeDstOrder` lack permission check

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	code/contracts/pay/payDB.sol#L148 code/contracts/pay/payDB.sol#L167	Acknowledged	Secure3

Code

```
148:     /// @notice called by payment nodes
149:     function executeDstOrderETH(

167:     /// @notice called by payment nodes
168:     function executeDstOrder(
```

Description

Secure3 : As described in the annotation(PayDB.sol#148,167), these two functions should only be called by payment nodes, but there is no permission check.

Recommendation

Secure3 : Add permission check in `_executeDstOrder` function to ensure `msg.sender` is payment node.

Client Response

Acknowledged. Order execution costs money hence there is no incentive for anyone to execute order for someone else. Also the node address will be different on different chains deployed, and there can be multiple nodes deployed on the same chain, hence the address binding is done on the "super node". To conclude, the consensus layer only ensures the order has been executed but does not enforce the execution address and creator's address are the same.

DAP-6: `VWManagerService::_resetOwner` needs to check if new owner is `address(0)`

Category	Severity	Code Reference	Status	Contributor
Logical	Low	code/contracts/vwallet/VWManagerService.sol#L79-L90	Fixed	Secure3

Code

```
79:     function _resetOwner(address wallet, address newOwner) internal returns (address
previousOwner){
80:         previousOwner = walletOwner[wallet];
81:
82:         walletOwner[wallet] = newOwner;
83:
84:         require(ownerWallet[newOwner] == address(0),"E4");
85:         ownerWallet[newOwner] = wallet;
86:
87:         if(previousOwner != address(0)){
88:             ownerWallet[previousOwner] = address(0);
89:         }
90:     }
```

Description

Secure3 : The new owner should not be `address(0)`, as it would result in losing control of the wallet.

Recommendation

Secure3 : Add check in `_resetOwner` function to ensure `newOwner` is now `address(0)`.

Client Response

Fixed. Added `newOwner != address(0)` in the check.

DAP-7: `code` should be bound to the `signer` (wallet owner) instead of the wallet to prevent signature replay attack

Category	Severity	Code Reference	Status	Contributor
Signature Forgery or Replay	Critical	code/contracts/vwallet/VWManager.sol#L41 code/contracts/vwallet/VWManagerService.sol#L14 code/contracts/vwallet/VWManagerService.sol#L33 code/contracts/vwallet/VWManagerService.sol#L53	Fixed	Secure3

Code

```
41:         require(result[msg.sender][code] == 0, "E2");

14:         require(result[wallet][code] == 0, "E2");

33:         require(result[wallet][code] == 0, "E2");

53:         require(result[wallet][code] == 0, "E2");
```

Description

Secure3 : `VWManager::verify()`, `VWManagerService::cancelTx()`, `VWManagerService::changeOwner()` and `VWManagerService::approveResetter()` are affected. To prevent signature replay attack, `nonce` is encoded in the `code`, and `code` is key of `result`. When calling above functions, `result` will be checked and changed before verifying signature:

```
require(result[wallet][code] == 0, "E2")
// ...
result[wallet][code] = 3;
SignLibrary.verify(walletOwner[wallet], domainSeparator[srcChain], dataHash, signature);
```

However, the signature signer(wallet owner) may own different wallets at different time periods. Signature can be replayed if a owner have a new wallet.

Recommendation

Secure3 : `code` should be bound to the `signer`(wallet owner) instead of the wallet. Considering change:

```
// Wallet => code => result
mapping(address => mapping(uint => uint)) public result;

require(result[wallet][code] == 0, "E2");

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

to

```
// owner => code => result
mapping(address => mapping(uint => uint)) public result;

require(result[walletOwner[wallet]][code] == 0, "E2");

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

Client Response

Fixed. the `code` is bound to the `signer` now.

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.