Competitive Security Assessment

dappOSP5

Aug 28th, 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	dappOSP5
Platform & Language	Solidity
Codebase	• https://github.com/DappOSDao/contracts-core/ • audit commit - 2a740d26708c52e3bbc8c71d9a1d810b5765ac3f • final commit - 99e3a0b2759d400e5bf837cefcd5924d4ed1b96e
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

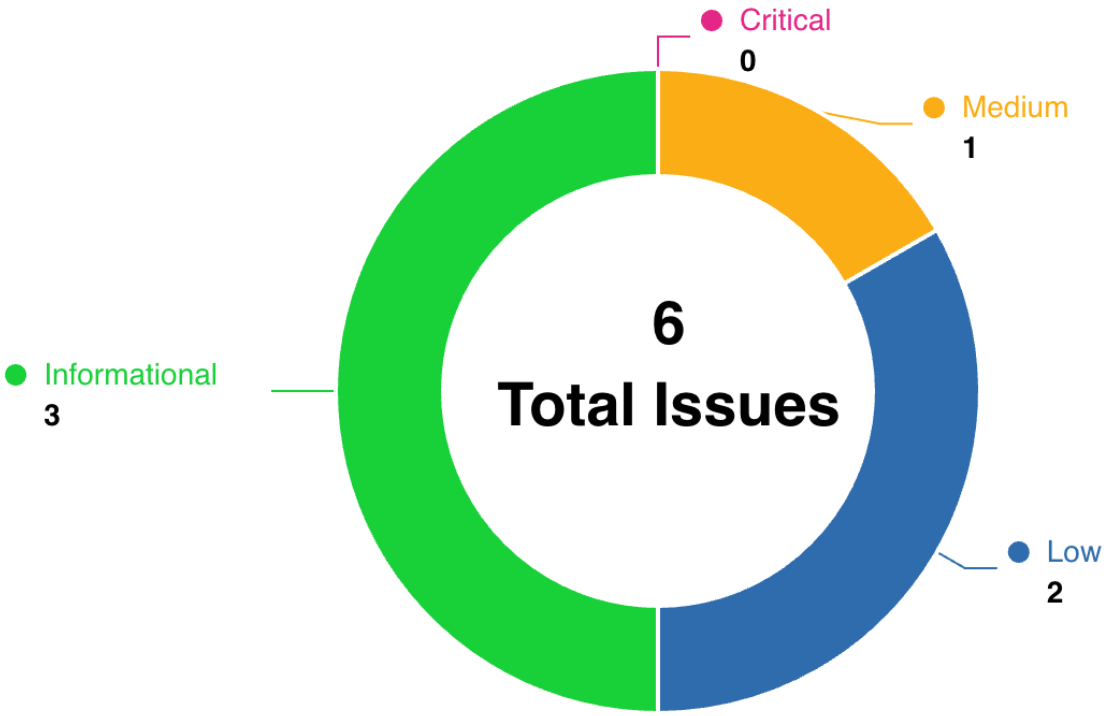
Code Vulnerability Review Summary

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

Audit Scope

File	SHA256 Hash
contracts/core/interfaces/IVWManager.sol	ee756eeacc5ab64a5487a21959f3dacca0b0a363c4f8cb62f2696e59f949af5d
contracts/core/interfaces/IVWManagerStorage.sol	8e679514773be47f08ff8cec149417b2b43715c5b9b19c82204e6c85a539a9fb
contracts/core/vwmanager/VWManager.sol	bfe96cebdbde7f83a880569fe0d6b3e0ec24f89d04a4d601395247adbc4fc82d
contracts/core/vwmanager/storage/VWManagerStorage.sol	0440aab31d8cea7d237a3f1f49b5fbe0e6f09c37fb0fff67b961ed76f75313b2

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
DAP-1	Potential front-run attack in the function `storeInfo()`	Logical	Medium	Fixed	Yaodao
DAP-2	Potential fail to verify in the function `verifyEIP1271Signature()`	Logical	Low	Declined	Yaodao
DAP-3	Inaccurate signature length check	Logical	Low	Fixed	Hacker007
DAP-4	Use calldata instead of memory	Gas Optimization	Informational	Fixed	Yaodao

DAP-5	**Lack of return value for the function `storeInfo`**	**Logical**	**Informational**	**Acknowledged**	**danielt**
DAP-6	**Gas Optimization in `VWManager.sol`**	**Gas Optimization**	**Informational**	**Fixed**	**Hacker007**

DAP-1:Potential front-run attack in the function `storeInfo()`

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	Yaodao

Code Reference

- code/contracts/core/vwmanager/VWManager.sol#L295-L309

```
295:function storeInfo(
296:        bytes memory info,
297:        bool willDelete
298:    ) external {
299:        if(info.length > 0){
300:            bytes32 infoHash = keccak256(info);
301:            if(eip1271Info[infoHash].length == 0){
302:                eip1271Info[infoHash] = info;
303:                emit InfoStored(infoHash, info);
304:                if(willDelete){
305:                    infoSender[infoHash] = msg.sender;
306:                }
307:            }
308:        }
309:    }
```

Description

Yaodao : According to the codes in the function `storeInfo()`, anyone can call this function to update the `infoSender[infoHash]` to be the `msg.sender`. So the attacker can call this function via front-run to update the info to be the address of attack. And then call the function `deleteInfo()` to delete the related information in the `eip1271Info` and `infoSender`. As a result, the real user can't verify the user's EIP1271 information.

```
    function storeInfo(
        bytes memory info,
        bool willDelete
    ) external {
        if(info.length > 0){
            bytes32 infoHash = keccak256(info);
            if(eip1271Info[infoHash].length == 0){
                eip1271Info[infoHash] = info;
                emit InfoStored(infoHash, info);
                if(willDelete){
                    infoSender[infoHash] = msg.sender;
                }
            }
        }
    }
```

Recommendation

Yaodao : Recommend deleting eip1271 information by the owner, rather than by the user's own operations.

Client Response

Fixed,The storage and usage of information in the storeInfo function will only occur within the same transaction. Therefore, if there is a race condition in transactions, the transaction that requires the information can still load the transactional data or utilize it directly by the user storing it.And we fix the problem, if some one want the msg to be store forever, he can rewrite the will delete info sender to address(0), so the info can be store forever

DAP-2:Potential fail to verify in the function `verifyEIP1271Signature()`

Category	Severity	Client Response	Contributor
Logical	Low	Declined	Yaodao

Code Reference

- code/contracts/core/vwmanager/VWManager.sol#L83-L103

```
83:if(uint8(bytes1(signature[64:65])) == 100){
84:                         (bytes32 orderHash, uint256 srcChain, bytes memory realSignature) = abi.d
ecode(
85:                             eip1271Info[bytes32(signature[:32])],
86:                             (bytes32,uint256,bytes)
87:                         );
88:                         (bytes32 dappDomainSeparator, bytes32 DAPPTYPEHASH, bytes32 SEPARATORTYPE
HASH) = abi.decode(
89:                             eip1271Info[bytes32(signature[32:64])],
90:                             (bytes32,bytes32,bytes32)
91:                         );
92:                         bytes32 signedDigest = _getSignedDigest(
93:                             digest,
94:                             srcChain,
95:                             orderHash,
96:                             dappDomainSeparator,
97:                             SEPARATORTYPEHASH,
98:                             DAPPTYPEHASH
99:                         );
100:                         isValid = SignLibrary.verifyEIP1271Signature(vwOwner, signedDigest, real
Signature);
101:                     } else {
102:                         isValid = SignLibrary.verifyEIP1271Signature(vwOwner, digest, signatur
e);
103:                     }
```

Description

Yaodao : According to the following codes, the logic in the `else(L83-103)` is used to deal with the signature that is not over 65 length.

```
    if(uint8(bytes1(signature[64:65])) == 100){
        (bytes32 orderHash, uint256 srcChain, bytes memory realSignature) = abi.decode(
            eip1271Info[bytes32(signature[:32])],
            (bytes32,uint256,bytes)
        );
        (bytes32 dappDomainSeparator, bytes32 DAPPTYPEHASH, bytes32 SEPARATORTYPEHASH) = abi.decode(
            eip1271Info[bytes32(signature[32:64])],
            (bytes32,bytes32,bytes32)
        );
        bytes32 signedDigest = _getSignedDigest(
            digest,
            srcChain,
            orderHash,
            dappDomainSeparator,
            SEPARATORTYPEHASH,
            DAPPTYPEHASH
        );
            isValid = SignLibrary.verifyEIP1271Signature(vwOwner, signedDigest, realSignature);
    } else {
        isValid = SignLibrary.verifyEIP1271Signature(vwOwner, digest, signature);
    }
```

In the `if(uint8(bytes1(signature[64:65])) == 100)` logic, the info is used for the logic designed by the client which is suitable for the signature info given by the user. Since normal 1271 signatures may also satisfy this if condition, and the info in the standardized EIP1271 signature is not suitable for the the logic designed by the client. As a result, the verify will always fail in this condition.

Recommendation

Yaodao : Recommend confirming the logic and redesiging it.

Client Response

Declined,In the case of owner being a non-address, the normal signature value for "v" would be either 27 or 28, not 100. Therefore, it will not interfere with normal signature process.

DAP-3:Inaccurate signature length check

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	Hacker007

Code Reference

- code/contracts/core/vwmanager/VWManager.sol#L73-L80
- code/contracts/core/vwmanager/VWManager.sol#L83

```
73:bytes32 signedDigest = _getSignedDigest(
74:                     digest,
75:                     uint256(bytes32(signature[:32])),
76:                     bytes32(signature[32:64]),
77:                     bytes32(signature[64:96]),
78:                     bytes32(signature[96:128]),
79:                     bytes32(signature[128:160])
80:                 );

83:if(uint8(bytes1(signature[64:65])) == 100){
```

Description

Hacker007 : The function `verifyEIP1271Signature()` does not check the signature length is valid before getting a signature slice. Getting slice beyond the length of the signature will be reverted.

```
            if(signature.length > 65){
                bytes32 signedDigest = _getSignedDigest(
                    digest,
                    uint256(bytes32(signature[:32])),
                    bytes32(signature[32:64]),
                    bytes32(signature[64:96]),
                    bytes32(signature[96:128]),
                    bytes32(signature[128:160])
                );
                isValid = SignLibrary.verifyEIP1271Signature(vwOwner, signedDigest, signature[16
0:]);
            } else {
                    if(uint8(bytes1(signature[64:65])) == 100){
```

Recommendation

Hacker007 : Check the signature length against the correct value before getting a slice.

Client Response

Fixed, change length check from 65 bytes to 65+160=225

DAP-4:Use calldata instead of memory

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	Yaodao

Code Reference

- code/contracts/core/vwmanager/VWManager.sol#L295-L298

```
295:function storeInfo(
296:        bytes memory info,
297:        bool willDelete
298:    ) external {
```

Description

Yaodao : It's better to use `calldata` instead of `memory` for function parameters that represent variables that will not be modified.

Recommendation

Yaodao : Recommend using calldata instead of memory to save gas.

Client Response

Fixed

DAP-5:Lack of return value for the function `storeInfo`

Category	Severity	Client Response	Contributor
Logical	Informational	Acknowledged	danielt

Code Reference

- code/contracts/core/vwmanager/VWManager.sol#L295-L309

```
295:function storeInfo(
296:        bytes memory info,
297:        bool willDelete
298:    ) external {
299:        if(info.length > 0){
300:            bytes32 infoHash = keccak256(info);
301:            if(eip1271Info[infoHash].length == 0){
302:                eip1271Info[infoHash] = info;
303:                emit InfoStored(infoHash, info);
304:                if(willDelete){
305:                    infoSender[infoHash] = msg.sender;
306:                }
307:            }
308:        }
309:    }
```

Description

danielt : The `storeInfo` function stores info for the `infoHash` into the mapping `eip1271Info`.
However, there is no result returned from the `storeInfo` function to the caller. The function may fail to store info if there is a conflict for the hash of different `info`.

Recommendation

danielt : Recommend returning a result, which stands for storing the info successfully or not.

Client Response

Acknowledged,When the length of info is greater than zero, the storage of info will always be successful. The usage of info also occurs within the same transaction, so there is no need to indicate a return value.

DAP-6:Gas Optimization in `VWManager.sol`

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	Hacker007

Code Reference

- code/contracts/core/vwmanager/VWManager.sol#L295-L298
- code/contracts/core/vwmanager/VWManager.sol#L315-L324

```
295:function storeInfo(
296:        bytes memory info,
297:        bool willDelete
298:    ) external {

315:function deleteInfo(
316:        bytes32 infoHash
317:    ) external {
318:        if(eip1271Info[infoHash].length > 0){
319:            require(msg.sender == infoSender[infoHash], "E33");
320:            delete eip1271Info[infoHash];
321:            delete infoSender[infoHash];
322:            emit InfoDeleted(infoHash);
323:        }
324:    }
```

Description

Hacker007 : Use calldata instead of memory for function parameters. Having function arguments use calldata instead of memory can save gas.

```
    function storeInfo(
        bytes memory info,
        bool willDelete
    ) external {
```

The function `storeInfo` implies that if infoSender[infoHash] is non-zero value, eip1271Info[infoHash] won't be a zero value,

```
            if(eip1271Info[infoHash].length == 0){
                eip1271Info[infoHash] = info;
                emit InfoStored(infoHash, info);
                if(willDelete){
                    infoSender[infoHash] = msg.sender;
                }
            }
```

Thus, it is redundant to check the length of eip1271Info[infoHash] is greater than zero before checking infoSender[infoHash] against `msg.sender` in the function `deleteInfo()`.

```
        if(eip1271Info[infoHash].length > 0){
            require(msg.sender == infoSender[infoHash], "E33");
            delete eip1271Info[infoHash];
            delete infoSender[infoHash];
            emit InfoDeleted(infoHash);
        }
```

Recommendation

Hacker007 : Change function arguments from memory to calldata. Remove the if statement in the function `deleteInfo()`.

```
    function deleteInfo(
        bytes32 infoHash
    ) external {
        require(msg.sender == infoSender[infoHash], "E33");
        delete eip1271Info[infoHash];
        delete infoSender[infoHash];
        emit InfoDeleted(infoHash);
    }
```

Client Response

Fixed, When the information has already been deleted, the length will be zero. Checking it can avoid duplicate deletions and save gas.

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.