DIDO Smart Contract

April 2024

SMART CONTRACT AUDIT REPORT

ExVul

www.exvul.com

Table of Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by **DIDO** to review smart contract implementation. The assessment was conducted in accordance with our systematic approach to evaluate potential security issues based upon customer requirement. The report provides detailed recommendations to resolve the issue and provide additional suggestions or recommendations for improvement.

High risk finding is primarily related to the token locked, owner permissions, code logic, etc. .

Medium risk finding is primarily related to the Gas limit .

Low risk finding is primarily related to the sign up code logic .

Informational risk finding is primarily related to the redundant code.

The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

1.1 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10] which is the gold standard in risk assessment using the following risk models:

- Likelihood: represents how likely a particular vulnerability is to be uncovered and exploited in the wild.
- Impact: measures the technical loss and business damage of a successful attack.
- Severity: determine the overall criticality of the risk.

Likelihood can be: High, Medium and Low and impact are categorized into for: High, Medium, Low, Informational. Severity is determined by likelihood and impact and can be classified into five categories accordingly, Critical, High, Medium, Low, Informational shown in table 1.1.

Likelihood	Informational	Low	Medium	High
High	*Informational*	Medium	High	Critical
Medium	*Informational*	Low	Medium	High
Low	*Informational*	Low	Low	Medium

IMPACT

Table 1.1 Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive assessment list carefully designed to identify known and impactful security issues. If our tool or analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy contracts on our private test environment and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.2.

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Code and business security testing: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Category	Assessment Item
Basic Coding Assessment	Apply Verification Control
	Authorization Access Control
	Forged Transfer Vulnerability
	Forged Transfer Notification
	Numeric Overflow
	Transaction Rollback Attack
	Transaction Block Stuffing Attack
	Soft Fail Attack
	Hard Fail Attack
	Abnormal Memo
	Abnormal Resource Consumption
	Secure Random Number
Advanced Source Code Scrutiny	Asset Security
	Cryptography Security
	Business Logic Review
	Source Code Functional Verification
	Account Authorization Control
	Sensitive Information Disclosure
	Circuit Breaker
	Blacklist Control

Category	Assessment Item
	System API Call Analysis
	Contract Deployment Consistency Check
Additional Recommendations	Semantic Consistency Checks
	Following Other Best Practices

Table 1.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2. FINDINGS OVERVIEW

2.1 Project Info And Contract Address

Project Name: DIDO

Audit Time: April1nd, 2024 – April3th, 2024

Language: solidity

File Name	Hash
DIDO.sol	383550F50702BA34E69F7E4F6201FD3ECA1C75ED
Staking.sol	37E2B47D38862225508426266ABB77A8FAA5CB1D

2.2 Summary

Severity	Found	
Critical	0	
High	3	🟥 🟥 🟥
Medium	1	🟨
Low	1	🟦
Informational	1	🟩

2.3 Key Findings

High risk finding is primarily related to the token locked, owner permissions, code logic, etc. .

Medium risk finding is primarily related to the Gas limit .

Low risk finding is primarily related to the sign up code logic .

Informational risk finding is primarily related to the redundant code.

ID	Severity	Findings Title	Status	Confirm
NVE-001	High	Stakers does not need to spend token when raise token is native token	Ignored	Confirmed
NVE-002	High	Project raised tokens are locked in the contract	Ignored	Confirmed
NVE-003	High	Owner could remove the project before end time	Ignored	Confirmed
NVE-004	Medium	Block gas limit could be exceeded causing DoS	Ignored	Confirmed

ID	Severity	Findings Title	Status	Confirm
NVE-005	Low	Misconfigured project sign token causes the sign up logic misbehaves	Ignored	Confirmed
NVE-006	Informational	Remove redundant information	Ignored	Confirmed

Table 2.1: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

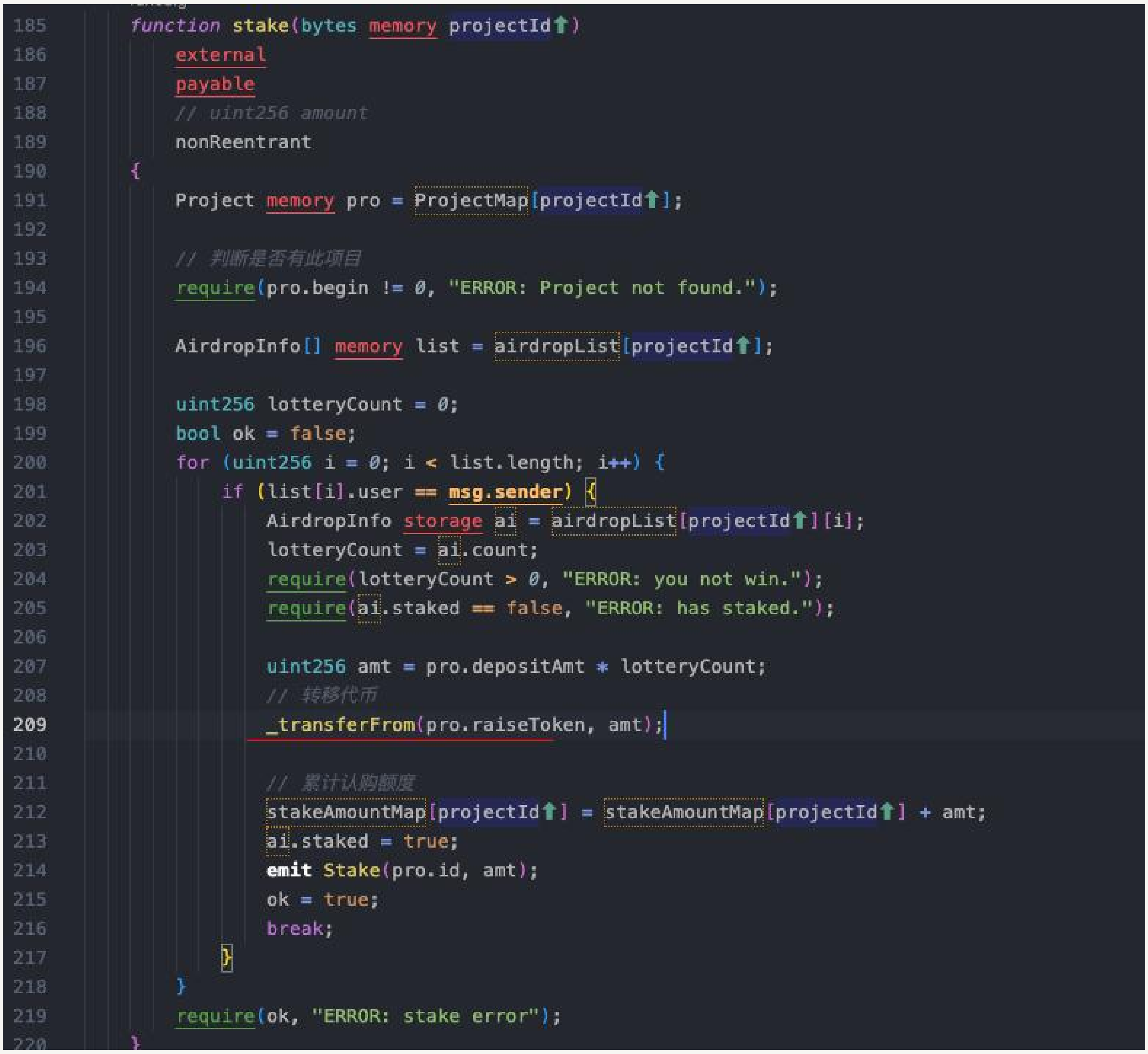
3.1 Stakers does not need to spend token when raise token is native token

ID:	NVE-001	**Location:**	DIDO.sol
Severity:	High	**Category:**	Business Issues
Likelihood:	Medium	**Impact:**	High

Description:

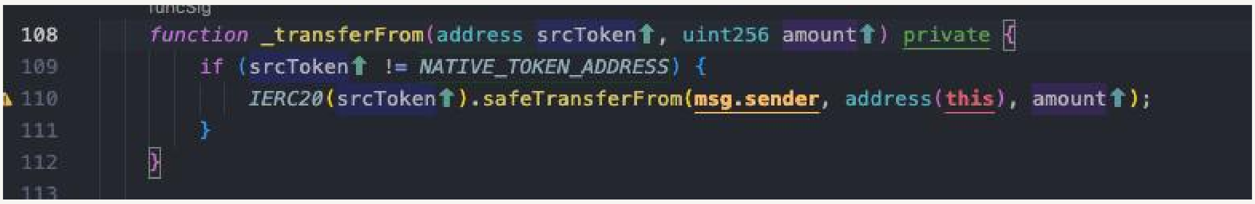
As shown in the figure below, the function `stake()` allows users to stake for a project given `projectId`. In the function, the users will have to stake the token `pro.raiseToken`

The issue happens when the project’s raise token is native token. In this case, the stakers won't have to spend the token to stake.



```
    function stake(bytes memory projectId)
        external
        payable
        // uint256 amount
        nonReentrant
    {
        Project memory pro = ProjectMap[projectId];

        // 判断是否有此项目
        require(pro.begin != 0, "ERROR: Project not found.");

        AirdropInfo[] memory list = airdropList[projectId];

        uint256 lotteryCount = 0;
        bool ok = false;
        for (uint256 i = 0; i < list.length; i++) {
            if (list[i].user == msg.sender) {
                AirdropInfo storage ai = airdropList[projectId][i];
                lotteryCount = ai.count;
                require(lotteryCount > 0, "ERROR: you not win.");
                require(ai.staked == false, "ERROR: has staked.");

                uint256 amt = pro.depositAmt * lotteryCount;
                // 转移代币
                _transferFrom(pro.raiseToken, amt);

                // 累计认购额度
                stakeAmountMap[projectId] = stakeAmountMap[projectId] + amt;
                ai.staked = true;
                emit Stake(pro.id, amt);
                ok = true;
                break;
            }
        }
        require(ok, "ERROR: stake error");
    }
```

Figure 3.1.1 DIDO.sol



```
function _transferFrom(address srcToken, uint256 amount) private {
    if (srcToken != NATIVE_TOKEN_ADDRESS) {
        IERC20(srcToken).safeTransferFrom(msg.sender, address(this), amount);
    }
}
```

Figure 3.1.2 DIDO.sol

Recommendations:

ExVul Web3 Labs recommends checking for `msg.value` in case the project's raise token is native token.

Result: Confirmed

Fix Result: Ignore

3.2 Project raised tokens are locked in the contract

ID:	NVE-002	**Location:**	DIDO.sol
Severity:	Medium	**Category:**	Business Issues
Likelihood:	High	**Impact:**	High

Description:

- According to the current contract implementation, there is no way that the project raise tokens could be transferred out of the contract.
- In case there are tokens accidentally transferred to the contract, OR there are remaining project tokens in the contract, these funds will get locked in the contract forever.

Recommendations:

ExVul Web3 Labs recommends adding a function to transfer the raised token out.

Result: Confirmed

Fix Result: Ignore

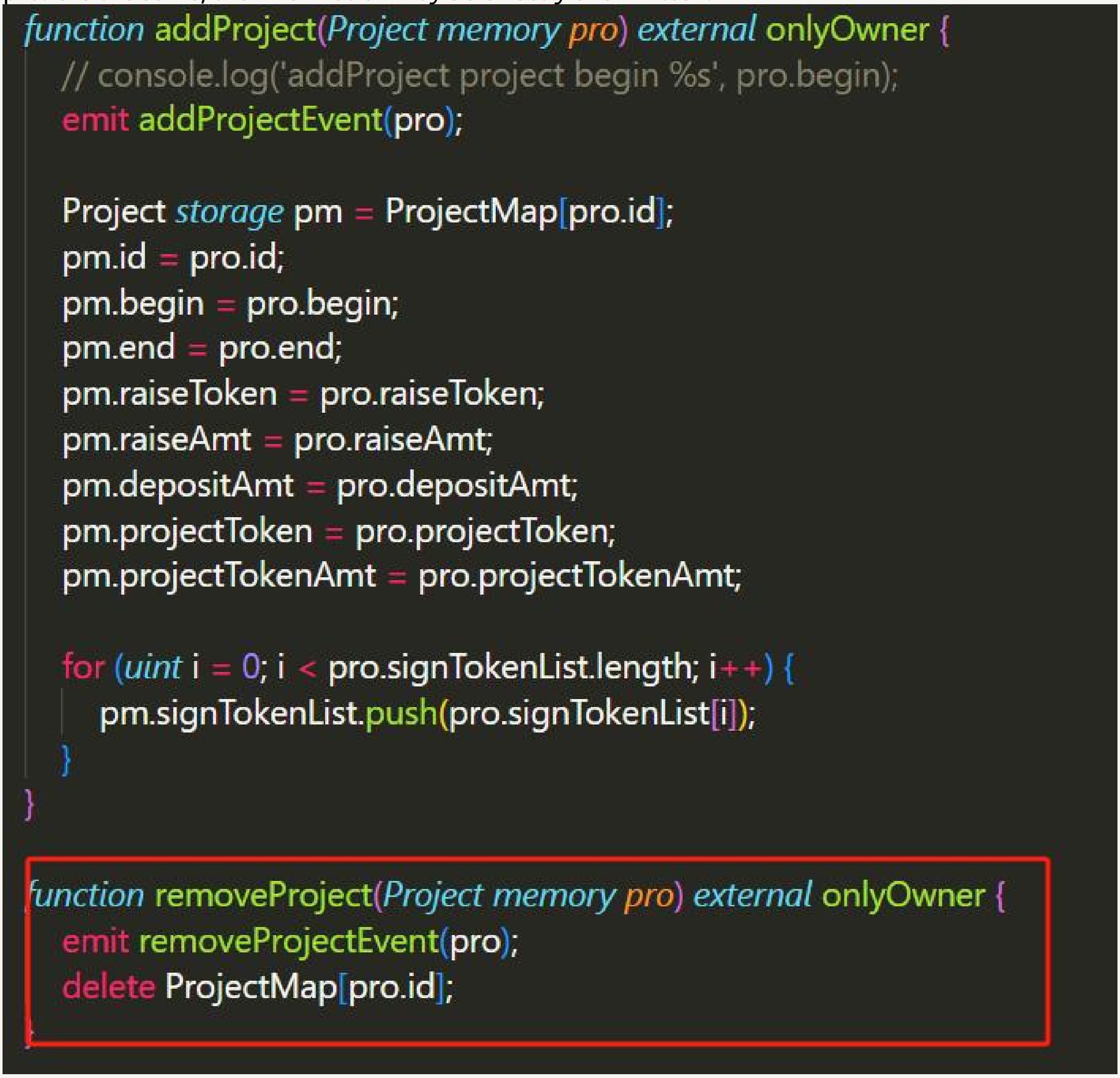
3.3 Owner could remove the project before end time

ID:	NVE-003	**Location:**	DIDO.sol
Severity:	High	**Category:**	Business Issues
Likelihood:	medium	**Impact:**	High

Description:

As shown in the figure below, the owner has the privilege to add projects and remove projects. In case the project is added and there are many stakers staked funds in the project, the malicious owner can remove the project. This will make the project stakers lost the staked funds.

At the same time, the current status of the project is not checked when adding the project. If the pro.id is the same, the information may be directly overwritten.



```
function addProject(Project memory pro) external onlyOwner {
    // console.log('addProject project begin %s', pro.begin);
    emit addProjectEvent(pro);

    Project storage pm = ProjectMap[pro.id];
    pm.id = pro.id;
    pm.begin = pro.begin;
    pm.end = pro.end;
    pm.raiseToken = pro.raiseToken;
    pm.raiseAmt = pro.raiseAmt;
    pm.depositAmt = pro.depositAmt;
    pm.projectToken = pro.projectToken;
    pm.projectTokenAmt = pro.projectTokenAmt;

    for (uint i = 0; i < pro.signTokenList.length; i++) {
        pm.signTokenList.push(pro.signTokenList[i]);
    }
}

function removeProject(Project memory pro) external onlyOwner {
    emit removeProjectEvent(pro);
    delete ProjectMap[pro.id];
}
```

Figure 3.3.1 DIDO.sol

Recommendations:

ExVul Web3 Labs recommends adding judgment on the current status of the project.

Result: Confirmed

Fix Result: Ignored

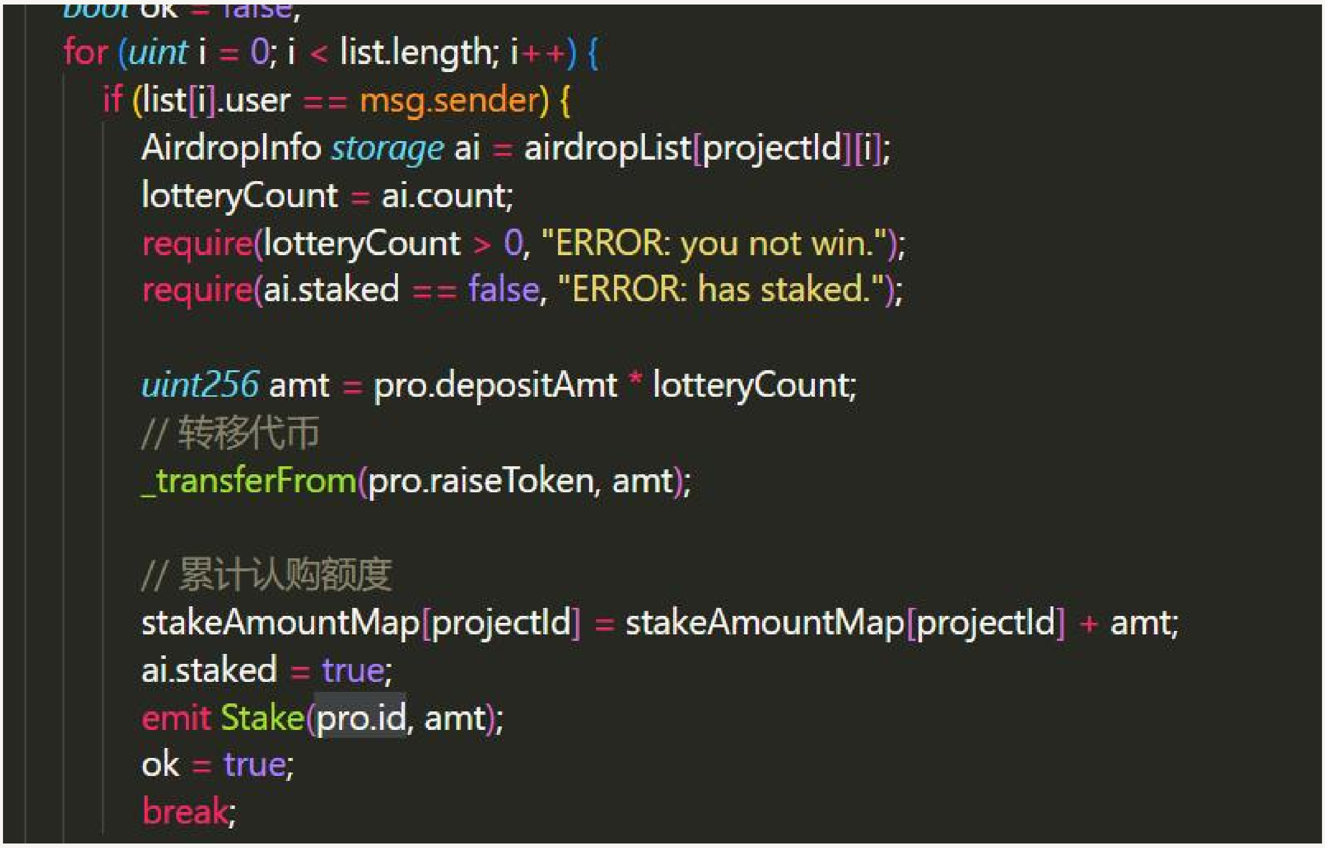
3.4 Block gas limit could be exceeded causing DoS

ID:	NVE-004	**Location:**	DIDO.sol
Severity:	Medium	**Category:**	Business Issues
Likelihood:	Medium	**Impact:**	Medium

Description:

As shown in the figure below, In the function `stake` and function `claim`, the caller is valid if only the caller is in the airdrop list.
The two functions’ implementation is gas expensive when:
Loading the airdrop list from storage to memory
Using for loop to find if `msg.sender` is in the list

The approach used in the implementation could cause the transaction to exceed block gas limit. When it happens, the functions are in DoS status. The extreme case for this is when the caller is the last user in the airdrop list.
.



```
bool ok = false;
for (uint i = 0; i < list.length; i++) {
    if (list[i].user == msg.sender) {
        AirdropInfo storage ai = airdropList[projectId][i];
        lotteryCount = ai.count;
        require(lotteryCount > 0, "ERROR: you not win.");
        require(ai.staked == false, "ERROR: has staked.");

        uint256 amt = pro.depositAmt * lotteryCount;
        // 转移代币
        _transferFrom(pro.raiseToken, amt);

        // 累计认购额度
        stakeAmountMap[projectId] = stakeAmountMap[projectId] + amt;
        ai.staked = true;
        emit Stake(pro.id, amt);
        ok = true;
        break;
```

Figure 3.4.1 DIDO.sol

Recommendations:

ExVul Web3 Labs recommends considering for loop depth.

Result: Confirmed

Fix Result: Ignored

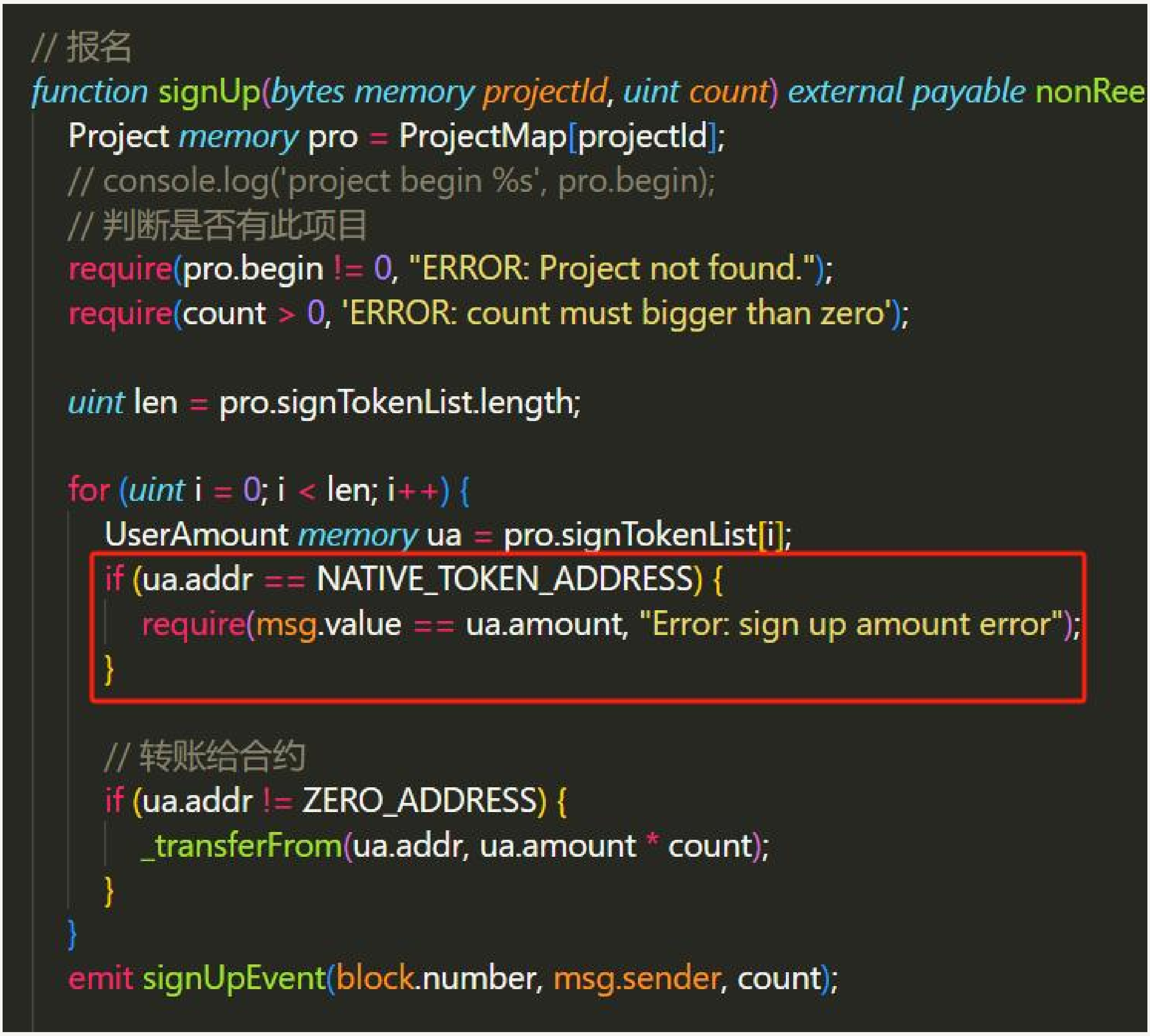
3.5 Misconfigured project sign token causes the sign up logic misbehaves

ID:	NVE-005	**Location:**	DIDO.sol
Severity:	Low	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Low

Description:

As shown in the figure below, in case the `signTokenList` is misconfigured by the owner, especially when the token address is duplicated in the list, the `signUp()` function will misbehave.
In case the native token is duplicated, the `msg.value` is used only once when there are many native token address in the list

.



```
// 报名
function signUp(bytes memory projectId, uint count) external payable nonRee
    Project memory pro = ProjectMap[projectId];
    // console.log('project begin %s', pro.begin);
    // 判断是否有此项目
    require(pro.begin != 0, "ERROR: Project not found.");
    require(count > 0, 'ERROR: count must bigger than zero');

    uint len = pro.signTokenList.length;

    for (uint i = 0; i < len; i++) {
        UserAmount memory ua = pro.signTokenList[i];
        if (ua.addr == NATIVE_TOKEN_ADDRESS) {
            require(msg.value == ua.amount, "Error: sign up amount error");
        }

        // 转账给合约
        if (ua.addr != ZERO_ADDRESS) {
            _transferFrom(ua.addr, ua.amount * count);
        }
    }
    emit signUpEvent(block.number, msg.sender, count);
```

Figure 3.5.1 DIDO.sol

Recommendations:

ExVul Web3 Labs recommends modifying code logic.

Result: Confirmed

Fix Result: Ignored

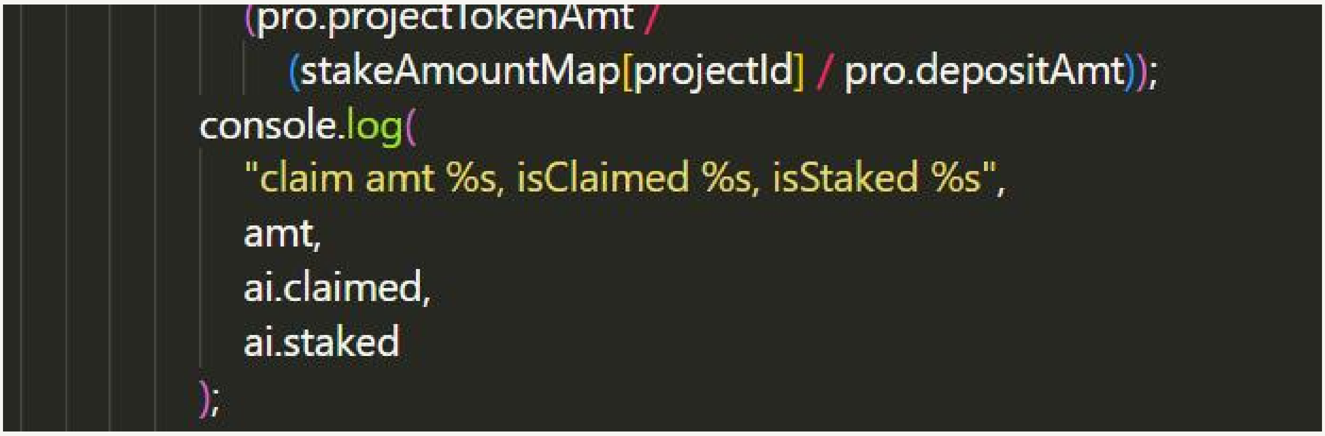
3.6 Remove redundant information

ID:	NVE-006	**Location:**	DIDO.sol
Severity:	Informational	**Category:**	Business Issues
Likelihood:	Informational	**Impact:**	Informational

Description:

As shown in the figure below, there are some redundant output information codes in the contract.

.



```
            (pro.projectTokenAmt /
                (stakeAmountMap[projectId] / pro.depositAmt));
        console.log(
            "claim amt %s, isClaimed %s, isStaked %s",
            amt,
            ai.claimed,
            ai.staked
        );
```

Figure 3.6.1 DIDO.sol

Recommendations:

ExVul Web3 Labs recommends deleting redundant code .

Result: Confirmed

Fix Result: Ignored

4. CONCLUSION

In this audit, we thoroughly analyzed **DIDO** smart contract implementation. The problems found are described and explained in detail in Section 3. The problems found in the audit have been communicated to the project leader. We therefore consider the audit result to be PASSED. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

5. APPENDIX

5.1 Basic Coding Assessment

5.1.1 Apply Verification Control

- Description: The security of apply verification
- Result: Not found
- Severity: Critical

5.1.2 Authorization Access Control

- Description: Permission checks for external integral functions
- Result: Not found
- Severity: Critical

5.1.3 Forged Transfer Vulnerability

- Description: Assess whether there is a forged transfer notification vulnerability in the contract
- Result: Not found
- Severity: Critical

5.1.4 Transaction Rollback Attack

- Description: Assess whether there is transaction rollback attack vulnerability in the contract.
- Result: Not found
- Severity: Critical

5.1.5 Transaction Block Stuffing Attack

- Description: Assess whether there is transaction blocking attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.6 Soft Fail Attack Assessment

- Description: Assess whether there is soft fail attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.7 Hard Fail Attack Assessment

- Description: Examine for hard fail attack vulnerability
- Result: Not found
- Severity: Critical

5.1.8 Abnormal Memo Assessment

- Description: Assess whether there is abnormal memo vulnerability in the contract.
- Result: Not found
- Severity: Critical

5.1.9 Abnormal Resource Consumption

- Description: Examine whether abnormal resource consumption in contract processing.
- Result: Not found
- Severity: Critical

5.1.10 Random Number Security

- Description: Examine whether the code uses insecure random number.
- Result: Not found
- Severity: Critical

5.2 Advanced Code Scrutiny

5.2.1 Cryptography Security

- Description: Examine for weakness in cryptograph implementation.
- Results: Not Found
- Severity: High

5.2.2 Account Permission Control

- Description: Examine permission control issue in the contract
- Results: Not Found
- Severity: Medium

5.2.3 Malicious Code Behavior

- Description: Examine whether sensitive behavior present in the code
- Results: Not found
- Severity: Medium

5.2.4 Sensitive Information Disclosure

- Description: Examine whether sensitive information disclosure issue present in the code.
- Result: Not found
- Severity: Medium

5.2.5 System API

- Description: Examine whether system API application issue present in the code
- Results: Not found
- Severity: Low

6. 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 Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without ExVul's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts ExVul to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. ExVul's position is that each company and individual are responsible for their own due diligence and continuous security. ExVul's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

7. REFERENCES

[1] MITRE. CWE- 191: Integer Underflow (Wrap or Wraparound).

https://cwe.mitre.org/data/ definitions/191.html.

[2] MITRE. CWE- 197: Numeric Truncation Error.

https://cwe.mitre.org/data/definitions/197. html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption.

https://cwe.mitre.org/data/ definitions/400.html.

[4] MITRE. CWE-440: Expected Behavior Violation.

https://cwe.mitre.org/data/definitions/440. html.

[5] MITRE. CWE-684: Protection Mechanism Failure.

https://cwe.mitre.org/data/definitions/ 693.html.

[6] MITRE. CWE CATEGORY: 7PK - Security Features.

https://cwe.mitre.org/data/definitions/ 254.html.

[7] MITRE. CWE CATEGORY: Behavioral Problems.

https://cwe.mitre.org/data/definitions/438. html.

[8] MITRE. CWE CATEGORY: Numeric Errors.

https://cwe.mitre.org/data/definitions/189.html.

[9] MITRE. CWE CATEGORY: Resource Management Errors.

https://cwe.mitre.org/data/ definitions/399.html.

[10] OWASP. Risk Rating Methodology.

https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

www.exvul.com

contact@exvul.com

@EXVULSEC

github.com/EXVUL-Sec

ExVul