Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	PAAWDNAH
Platform:	Ethereum
Language:	Solidity
Date:	May 25th, 2023

Table of contents

Introduction 4
Project Background 4
Audit Scope 4
Claimed Smart Contract Features 5
Audit Summary 6
Technical Quick Stats 7
Code Quality 8
Documentation 8
Use of Dependencies 8
AS-IS overview 9
Severity Definitions 11
Audit Findings 12
Conclusion 29
Our Methodology 30
Disclaimers 32
Appendix
- Code Flow Diagram 33
- Slither Results Log 34
- Solidity static analysis 36
- Solhint Linter 38

`

Introduction

EtherAuthority was contracted by the PAAWDNAH team to perform the Security audit of the PAAWDNAH protocol smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on May 25th, 2023.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- PAAWDNAH is a smart contract having functions like pause, unpause, withdraw, deposit, revertState, etc. .
- PAAWDNAH contract inherits the AccessControlUpgradeable, ReentrancyGuardUpgradeable, SafeMathUpgradeable, IERC20Upgradeable, PausableUpgradeable, Initializable standard smart contracts from the OpenZeppelin library.
- These OpenZeppelin contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

Audit scope

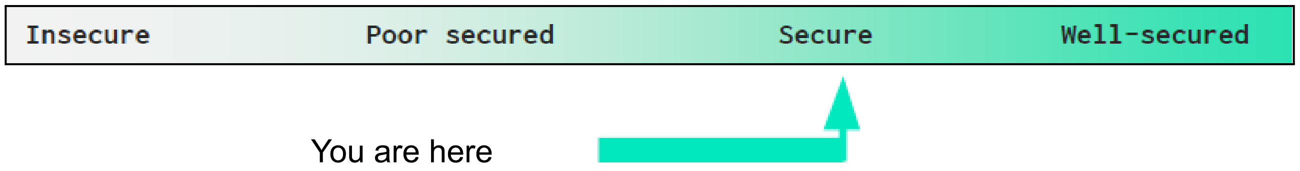
Name	**Code Review and Security Analysis Report for PAAWDNAH Token Smart Contract**
Platform	**Ethereum / Solidity**
File	PAAWDNAH.sol
File MD5 Hash	5D68E919E52C109671F4806E3C0DD9E6
Updated File MD5 Hash	A297160D58C01585F6D4802CB478ECBD
Audit Date	May 25th, 2023
Updated Audit Date	June 1st, 2023

Claimed Smart Contract Features

Claimed Feature Detail	**Our Observation**
Fee detail: ● Fee Rate: 7%	**YES, This is valid. Owner wallet's private key must be handled very securely. Because if that is compromised, then it will create problems.**
<u>Owner/Admin role has control over following functions:</u> ● Add funds to the withdrawal wallet. ● Add funds to the backup wallet. ● Update Deposit amount. ● Revert state if no new deposits are made in 2 weeks. ● Pause / Unpause the contract.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **"Secured".** This token contract does contain owner control, which does not make it fully decentralized.



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 11 high, 1 medium and 4 low and 9 very low level issues.
We confirm that 11 high, 1 medium, 3 low and 8 informational severity issues are fixed in the revised smart contract code.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Moderated
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	"Out of Gas" Issue	Passed
	High consumption 'for/while' loop	Passed
	High consumption 'storage' storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	"Short Address" Attack	Passed
	"Double Spend" Attack	Passed

Overall Audit Result: PASSED

Code Quality

This audit scope has 1 smart contract. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the PAAWDNAH Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the PAAWDNAH Token.

The PAAWDNAH team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is preferred as it increases the readability.

Documentation

We were given a PAAWDNAH Token smart contract code in the form of a file. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **well** commented. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initializer	modifier	Passed	No Issue
3	reinitializer	modifier	Passed	No Issue
4	onlyInitializing	modifier	Passed	No Issue
5	_disableInitializers	internal	Passed	No Issue
6	__AccessControl_init	internal	access only Initializing	No Issue
7	__AccessControl_init_unchained	internal	access only Initializing	No Issue
8	onlyRole	modifier	Passed	No Issue
9	supportsInterface	read	Passed	No Issue
10	hasRole	read	Passed	No Issue
11	_checkRole	internal	Passed	No Issue
12	_checkRole	internal	Passed	No Issue
13	getRoleAdmin	read	Passed	No Issue
14	grantRole	write	access only Role	No Issue
15	revokeRole	write	access only Role	No Issue
16	renounceRole	write	Passed	No Issue
17	_setupRole	internal	Passed	No Issue
18	_setRoleAdmin	internal	Passed	No Issue
19	_grantRole	internal	Passed	No Issue
20	_revokeRole	internal	Passed	No Issue
21	__ReentrancyGuard_init	internal	access only Initializing	No Issue
22	__ReentrancyGuard_init_unchained	internal	access only Initializing	No Issue
23	nonReentrant	modifier	Passed	No Issue
24	_nonReentrantBefore	write	Passed	No Issue
25	_nonReentrantAfter	write	Passed	No Issue
26	_reentrancyGuardEntered	internal	Passed	No Issue
27	__Pausable_init	internal	access only Initializing	No Issue
28	__Pausable_init_unchained	internal	access only Initializing	No Issue
29	whenNotPaused	modifier	Passed	No Issue
30	whenPaused	modifier	Passed	No Issue
31	paused	read	Passed	No Issue
32	_requireNotPaused	internal	Passed	No Issue
33	_requirePaused	internal	Passed	No Issue
34	_pause	internal	Passed	No Issue
35	_unpause	internal	Passed	No Issue
36	initialize	write	Passed	No Issue
37	_withdraw	internal	Passed	Removed
38	deposit	external	Passed	No Issue
39	withdraw	external	Withdrawal Issue (Low severity)	Refer Audit findings

40	revertFunds	write	access only Role	Removed
41	setMinimumBackupWalletBal ance	external	Passed	Removed
42	getDepositorDeposits	external	Passed	Removed
43	getTotalDepositAmount	external	Passed	Removed
44	pause	external	access only Role	No Issue
45	unpause	external	access only Role	No Issue
46	setMaintenanceFeeRate	external	Passed	Removed
47	setBackupPlanFeeRate	external	Passed	Removed
48	setQueFeeRate	external	Passed	Removed
49	withdrawFees	external	access only Role	Removed
50	updateMaintenanceFee	external	Passed	Removed
51	depositBackupWallet	external	Passed	Removed
52	depositWithdrawalWallet	external	Passed	Removed
53	withdrawMaintenanceFee	external	Passed	Removed
54	removeFirstDepositorFromQu eue	internal	Passed	No Issue
55	setMaxDepositors	write	Passed	No Issue
56	addFundsToWithdrawalWallet	write	Passed	No Issue
57	addFundsToBackupWallet	write	access only Role	No Issue
58	setDepositAmount	write	access only Role	No Issue
59	getDepositorPositionInQueue	external	Passed	No Issue
60	revertState	write	access only Role	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

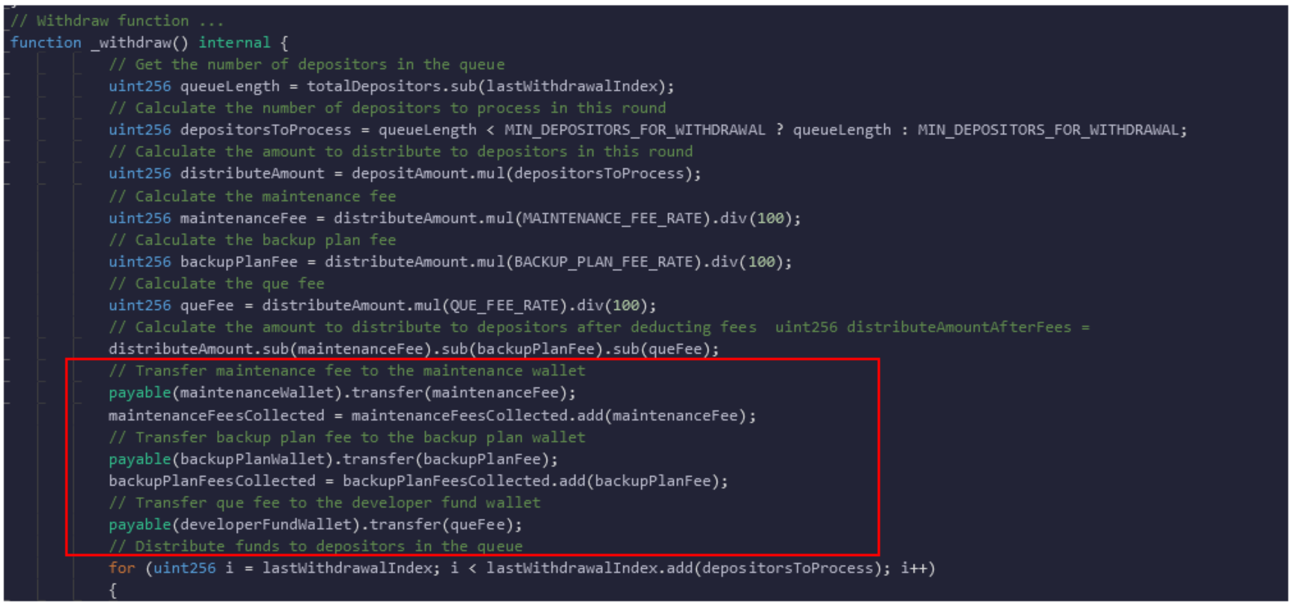
Critical Severity

No Critical severity vulnerabilities were found.

High Severity

(1) Not able to transfer maintenanceFee, backupPlanFee and queFee in _widthdraw function when totalDepositors reaches MIN_DEPOSITORS_FOR_WITHDRAWAL:

Function: _withdraw()



```
// Withdraw function ...
function _withdraw() internal {
        // Get the number of depositors in the queue
        uint256 queueLength = totalDepositors.sub(lastWithdrawalIndex);
        // Calculate the number of depositors to process in this round
        uint256 depositorsToProcess = queueLength < MIN_DEPOSITORS_FOR_WITHDRAWAL ? queueLength : MIN_DEPOSITORS_FOR_WITHDRAWAL;
        // Calculate the amount to distribute to depositors in this round
        uint256 distributeAmount = depositAmount.mul(depositorsToProcess);
        // Calculate the maintenance fee
        uint256 maintenanceFee = distributeAmount.mul(MAINTENANCE_FEE_RATE).div(100);
        // Calculate the backup plan fee
        uint256 backupPlanFee = distributeAmount.mul(BACKUP_PLAN_FEE_RATE).div(100);
        // Calculate the que fee
        uint256 queFee = distributeAmount.mul(QUE_FEE_RATE).div(100);
        // Calculate the amount to distribute to depositors after deducting fees  uint256 distributeAmountAfterFees =
        distributeAmount.sub(maintenanceFee).sub(backupPlanFee).sub(queFee);
        // Transfer maintenance fee to the maintenance wallet
        payable(maintenanceWallet).transfer(maintenanceFee);
        maintenanceFeesCollected = maintenanceFeesCollected.add(maintenanceFee);
        // Transfer backup plan fee to the backup plan wallet
        payable(backupPlanWallet).transfer(backupPlanFee);
        backupPlanFeesCollected = backupPlanFeesCollected.add(backupPlanFee);
        // Transfer que fee to the developer fund wallet
        payable(developerFundWallet).transfer(queFee);
        // Distribute funds to depositors in the queue
        for (uint256 i = lastWithdrawalIndex; i < lastWithdrawalIndex.add(depositorsToProcess); i++)
        {
```

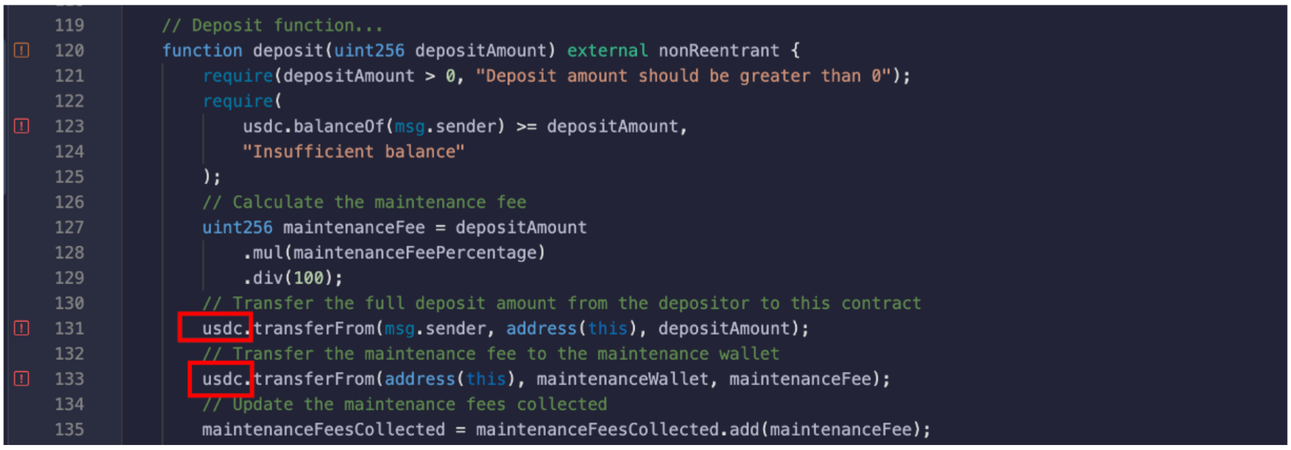
Not able to transfer maintenanceFee, backupPlanFee and queFee in _widthdraw function when totalDepositors deposits reach MIN_DEPOSITORS_FOR_WITHDRAWAL.
The contract doesn't have enough coins to transfer fees to respective wallets, and the contract doesn't implement a receive or fallback function to receive coins.

Resolution: Suggest checking this logic.

Status: This is fixed in the revised smart contract code.

(2) Used the wrong variable:

Function: Deposit()



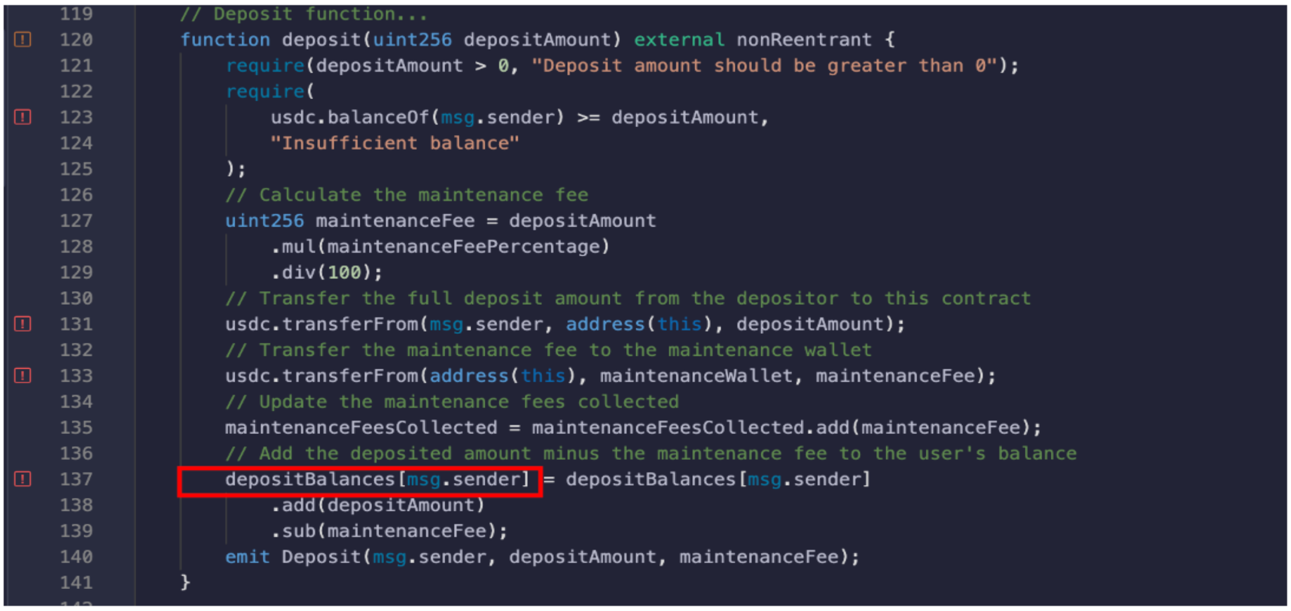
```
    // Deposit function...
    function deposit(uint256 depositAmount) external nonReentrant {
        require(depositAmount > 0, "Deposit amount should be greater than 0");
        require(
            usdc.balanceOf(msg.sender) >= depositAmount,
            "Insufficient balance"
        );
        // Calculate the maintenance fee
        uint256 maintenanceFee = depositAmount
            .mul(maintenanceFeePercentage)
            .div(100);
        // Transfer the full deposit amount from the depositor to this contract
        usdc.transferFrom(msg.sender, address(this), depositAmount);
        // Transfer the maintenance fee to the maintenance wallet
        usdc.transferFrom(address(this), maintenanceWallet, maintenanceFee);
        // Update the maintenance fees collected
        maintenanceFeesCollected = maintenanceFeesCollected.add(maintenanceFee);
```

The USDC variable is not correct, it should be USDC which is newly introduced.

Status: This is fixed in the revised smart contract code.

(3) Undeclared variable:

Function: Deposit()



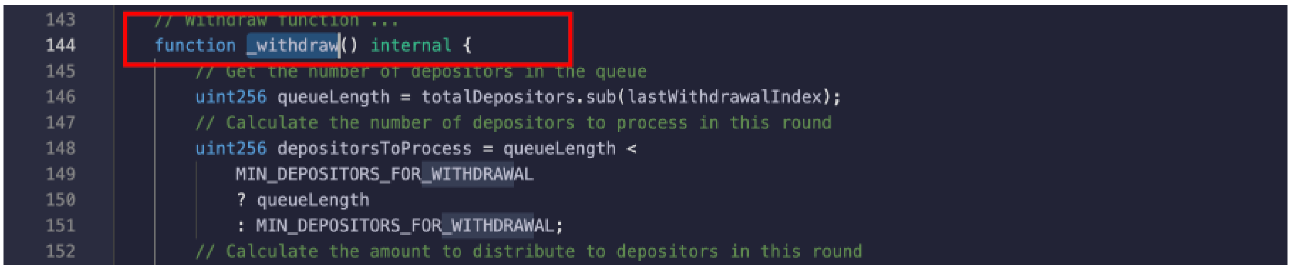
```
    // Deposit function...
    function deposit(uint256 depositAmount) external nonReentrant {
        require(depositAmount > 0, "Deposit amount should be greater than 0");
        require(
            usdc.balanceOf(msg.sender) >= depositAmount,
            "Insufficient balance"
        );
        // Calculate the maintenance fee
        uint256 maintenanceFee = depositAmount
            .mul(maintenanceFeePercentage)
            .div(100);
        // Transfer the full deposit amount from the depositor to this contract
        usdc.transferFrom(msg.sender, address(this), depositAmount);
        // Transfer the maintenance fee to the maintenance wallet
        usdc.transferFrom(address(this), maintenanceWallet, maintenanceFee);
        // Update the maintenance fees collected
        maintenanceFeesCollected = maintenanceFeesCollected.add(maintenanceFee);
        // Add the deposited amount minus the maintenance fee to the user's balance
        depositBalances[msg.sender] = depositBalances[msg.sender]
            .add(depositAmount)
            .sub(maintenanceFee);
        emit Deposit(msg.sender, depositAmount, maintenanceFee);
    }
```

Variable used without declaration.

Status: This is fixed in the revised smart contract code.

(4) Not able to withdraw funds, user deposit is locked inside contract:

Function: _withdraw()



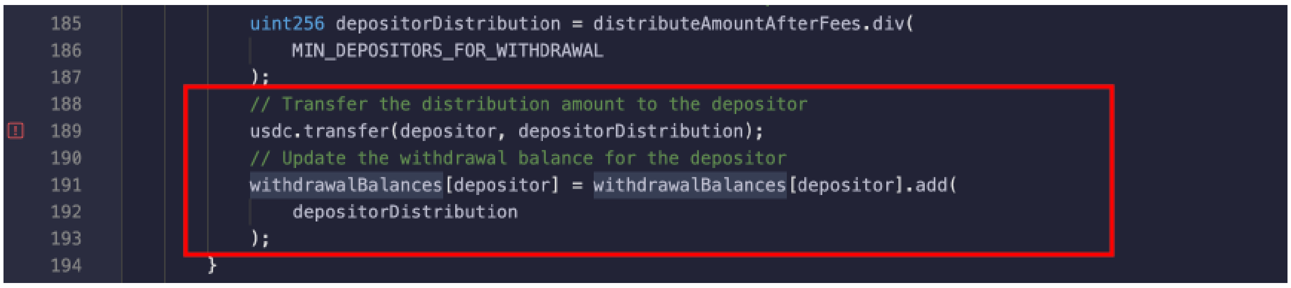
```
    // Withdraw function ...
    function _withdraw() internal {
        // Get the number of depositors in the queue
        uint256 queueLength = totalDepositors.sub(lastWithdrawalIndex);
        // Calculate the number of depositors to process in this round
        uint256 depositorsToProcess = queueLength <
            MIN_DEPOSITORS_FOR_WITHDRAWAL
            ? queueLength
            : MIN_DEPOSITORS_FOR_WITHDRAWAL;
        // Calculate the amount to distribute to depositors in this round
```

The _withdraw function is nowhere called from the contract, and it's marked as internal, so the user cannot withdraw funds.

Status: This is fixed in the revised smart contract code.

(5) After transferring the deposit amount among the depositors inside the _withdraw function, again maintaining the deposit amount against the user using withdrawalBalances:

Function: _withdraw()



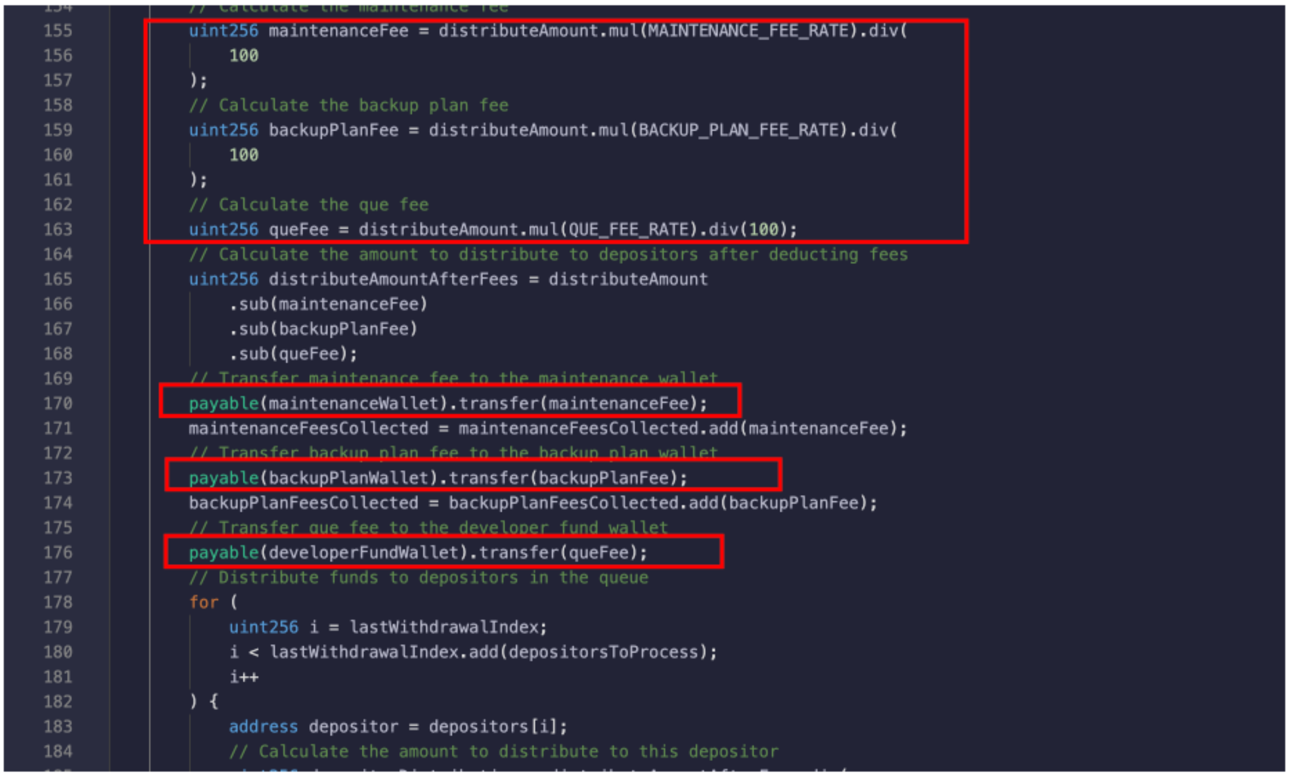
```
            uint256 depositorDistribution = distributeAmountAfterFees.div(
                MIN_DEPOSITORS_FOR_WITHDRAWAL
            );
            // Transfer the distribution amount to the depositor
            usdc.transfer(depositor, depositorDistribution);
            // Update the withdrawal balance for the depositor
            withdrawalBalances[depositor] = withdrawalBalances[depositor].add(
                depositorDistribution
            );
        }
```

After transferring the deposit amount among the depositors inside the _withdraw function, adding the withdrawal amount against the user using withdrawalBalances.

Status: This is fixed in the revised smart contract code.

(6) In withdraw function calculating 3 types of fees:

Function: withdraw()



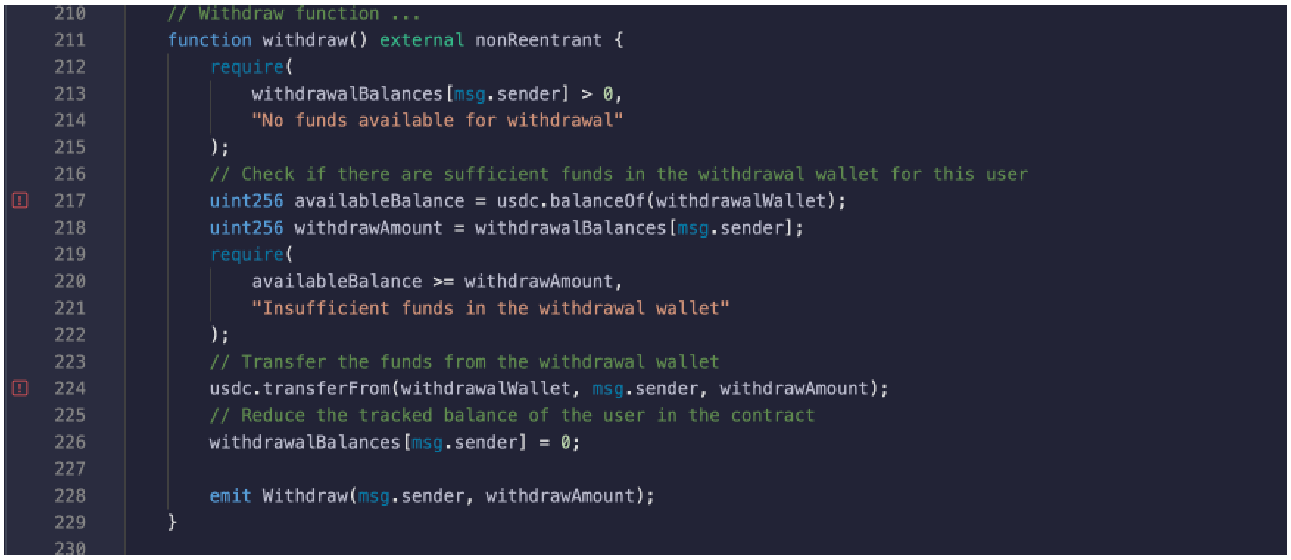
```
        uint256 maintenanceFee = distributeAmount.mul(MAINTENANCE_FEE_RATE).div(
            100
        );
        // Calculate the backup plan fee
        uint256 backupPlanFee = distributeAmount.mul(BACKUP_PLAN_FEE_RATE).div(
            100
        );
        // Calculate the que fee
        uint256 queFee = distributeAmount.mul(QUE_FEE_RATE).div(100);
        // Calculate the amount to distribute to depositors after deducting fees
        uint256 distributeAmountAfterFees = distributeAmount
            .sub(maintenanceFee)
            .sub(backupPlanFee)
            .sub(queFee);
        // Transfer maintenance fee to the maintenance wallet
        payable(maintenanceWallet).transfer(maintenanceFee);
        maintenanceFeesCollected = maintenanceFeesCollected.add(maintenanceFee);
        // Transfer backup plan fee to the backup plan wallet
        payable(backupPlanWallet).transfer(backupPlanFee);
        backupPlanFeesCollected = backupPlanFeesCollected.add(backupPlanFee);
        // Transfer que fee to the developer fund wallet
        payable(developerFundWallet).transfer(queFee);
        // Distribute funds to depositors in the queue
        for (
            uint256 i = lastWithdrawalIndex;
            i < lastWithdrawalIndex.add(depositorsToProcess);
            i++
        ) {
            address depositor = depositors[i];
            // Calculate the amount to distribute to this depositor
```

In the withdrawal function calculating 3 types of fees maintenance, backup and que fee, please check this.and after calculation of fees try to transfer native coins to respective wallet instead of usdc token.

Status: This is fixed in the revised smart contract code.

(7) Transferring the deposit amount twice:

Function: withdraw()



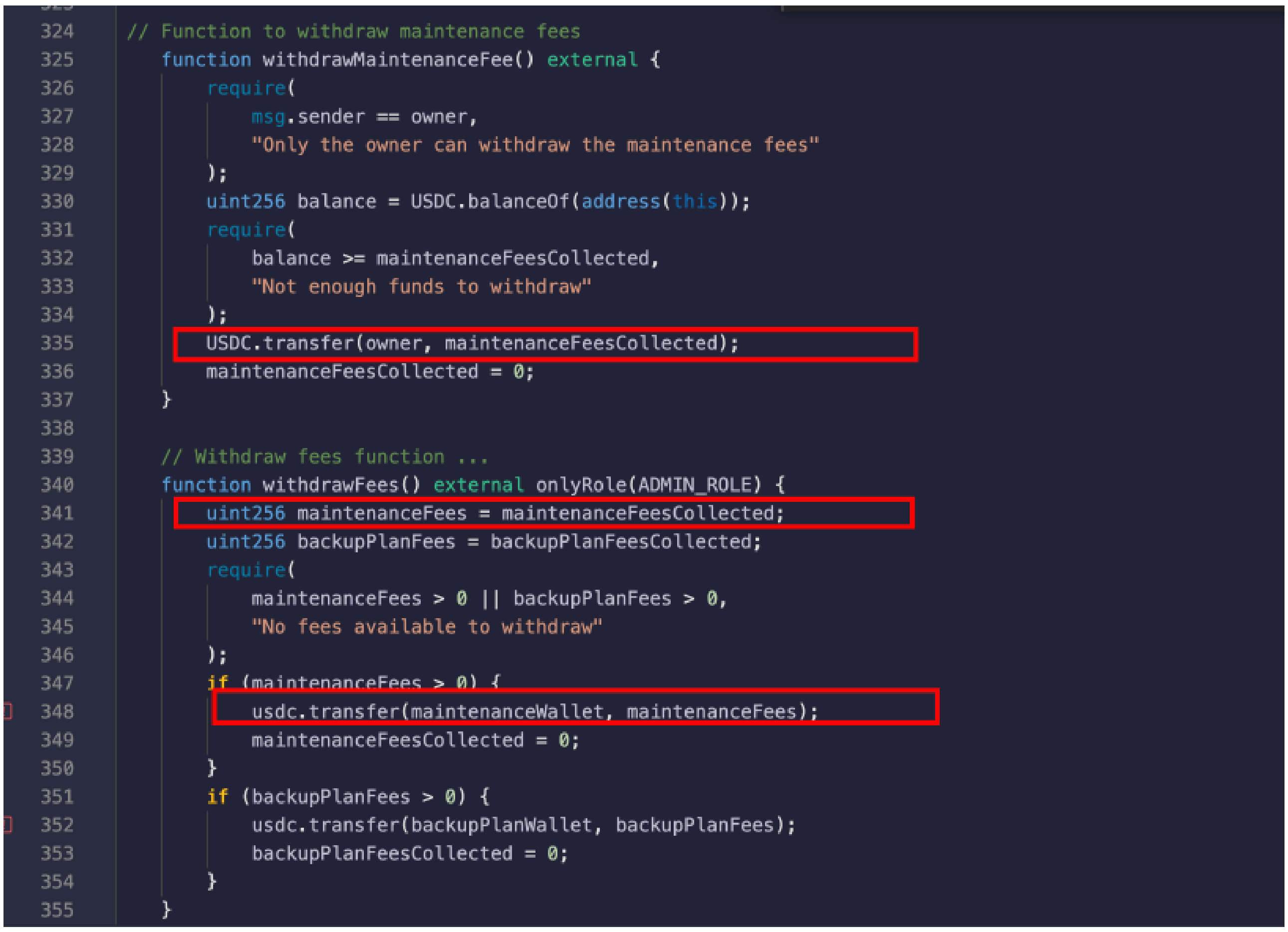
```
    // Withdraw function ...
    function withdraw() external nonReentrant {
        require(
            withdrawalBalances[msg.sender] > 0,
            "No funds available for withdrawal"
        );
        // Check if there are sufficient funds in the withdrawal wallet for this user
        uint256 availableBalance = usdc.balanceOf(withdrawalWallet);
        uint256 withdrawAmount = withdrawalBalances[msg.sender];
        require(
            availableBalance >= withdrawAmount,
            "Insufficient funds in the withdrawal wallet"
        );
        // Transfer the funds from the withdrawal wallet
        usdc.transferFrom(withdrawalWallet, msg.sender, withdrawAmount);
        // Reduce the tracked balance of the user in the contract
        withdrawalBalances[msg.sender] = 0;

        emit Withdraw(msg.sender, withdrawAmount);
    }
```

The withdrawal balance is already transferred to the depositors from _withdraw function, but again in the withdraw() function, transferring it twice.

Status: This is fixed in the revised smart contract code.

(8) Collecting maintenance Fees total 4 times in entire contract:

Function: withdrawMaintenanceFee()



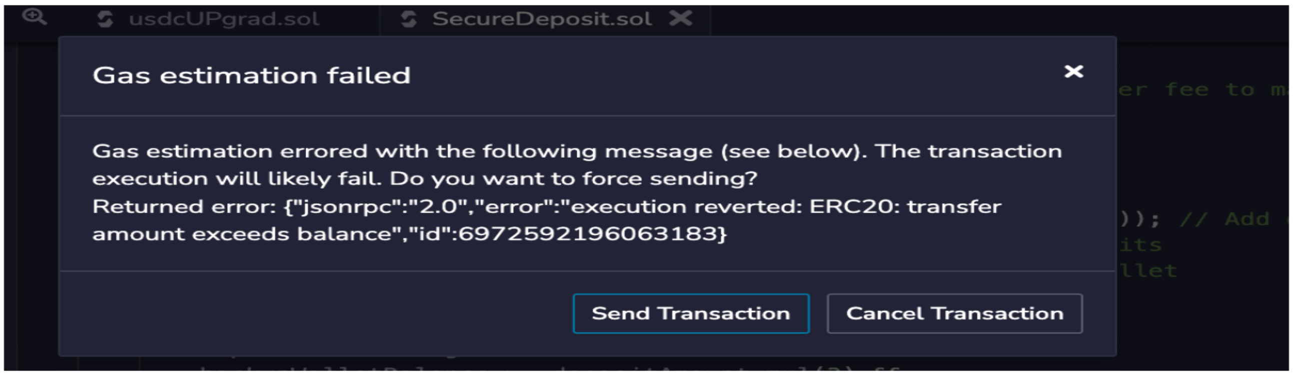
```
// Function to withdraw maintenance fees
function withdrawMaintenanceFee() external {
    require(
        msg.sender == owner,
        "Only the owner can withdraw the maintenance fees"
    );
    uint256 balance = USDC.balanceOf(address(this));
    require(
        balance >= maintenanceFeesCollected,
        "Not enough funds to withdraw"
    );
    USDC.transfer(owner, maintenanceFeesCollected);
    maintenanceFeesCollected = 0;
}

// Withdraw fees function ...
function withdrawFees() external onlyRole(ADMIN_ROLE) {
    uint256 maintenanceFees = maintenanceFeesCollected;
    uint256 backupPlanFees = backupPlanFeesCollected;
    require(
        maintenanceFees > 0 || backupPlanFees > 0,
        "No fees available to withdraw"
    );
    if (maintenanceFees > 0) {
        usdc.transfer(maintenanceWallet, maintenanceFees);
        maintenanceFeesCollected = 0;
    }
    if (backupPlanFees > 0) {
        usdc.transfer(backupPlanWallet, backupPlanFees);
        backupPlanFeesCollected = 0;
    }
}
```

Collecting maintenance fee again in the withdrawMaintenanceFee and withdrawFees, but it has already been collected in the deposit function.

Status: This is fixed in the revised smart contract code.

(9) Not able to deposit more than 4 times:

Error:



Gas estimation failed

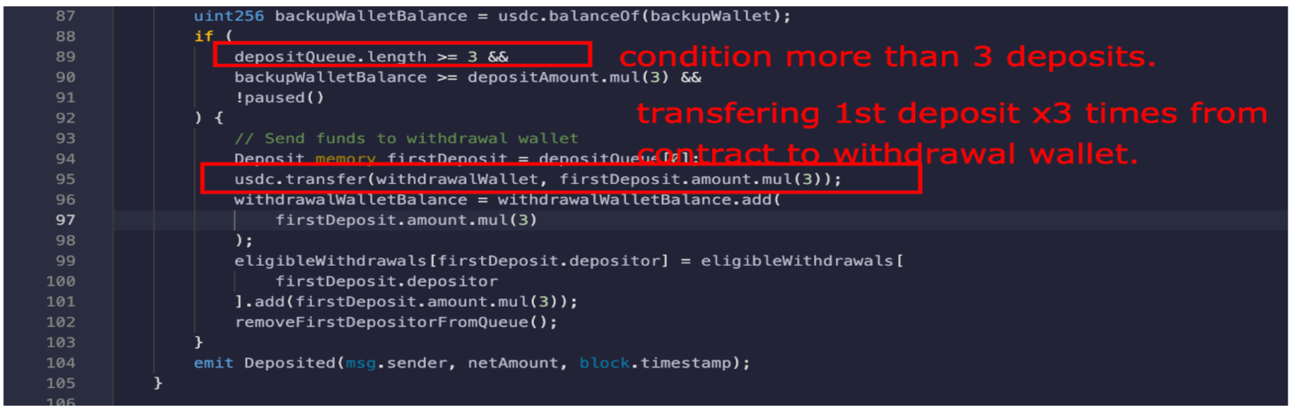
Gas estimation errored with the following message (see below). The transaction execution will likely fail. Do you want to force sending?
Returned error: {"jsonrpc":"2.0","error":"execution reverted: ERC20: transfer amount exceeds balance","id":6972592196063183}

1. Set the deposit amount to 1 USDC.
2. Deposit 4 times with 1 USDC.
3. Try to deposit 5 times.
4. not able to deposit more than 4 times.

<u>**The reason is:**</u>

1. On the 4th deposit the logic is: From 1st 3 deposits(1 USDC in each deposit) after deducting fee(7%) netamount = 0.93 x 3= 2.79 USDC is accumulated in the contract.
2. For the 4th deposit it will enter if condition(refer below code snippet) its withdrawing 1st deposited amount which is 0.93 x3 = 2.79 USDC from contract to withdrawal wallet.
3. So for the 5th deposit again it will enter if condition and hence no sufficient balance in contract for auto withdrawal so not able to deposit.

Function: deposit()



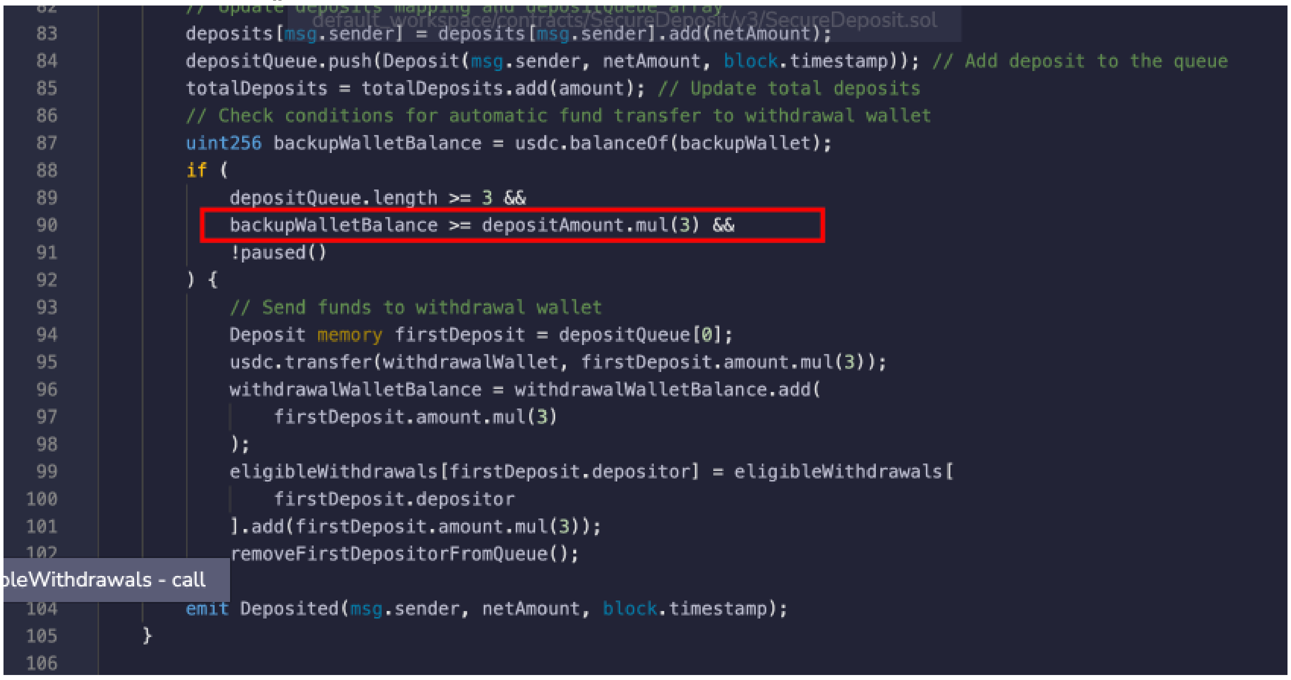
```
        uint256 backupWalletBalance = usdc.balanceOf(backupWallet);
        if (
            depositQueue.length >= 3 &&
            backupWalletBalance >= depositAmount.mul(3) &&
            !paused()
        ) {
            // Send funds to withdrawal wallet
            Deposit memory firstDeposit = depositQueue[0];
            usdc.transfer(withdrawalWallet, firstDeposit.amount.mul(3));
            withdrawalWalletBalance = withdrawalWalletBalance.add(
                firstDeposit.amount.mul(3)
            );
            eligibleWithdrawals[firstDeposit.depositor] = eligibleWithdrawals[
                firstDeposit.depositor
            ].add(firstDeposit.amount.mul(3));
            removeFirstDepositorFromQueue();
        }
        emit Deposited(msg.sender, netAmount, block.timestamp);
    }
```

condition more than 3 deposits.
transfering 1st deposit x3 times from contract to withdrawal wallet.

Status: This is fixed in the revised smart contract code.

(10) Contract will be freezed for 2 weeks at certain scenarios:

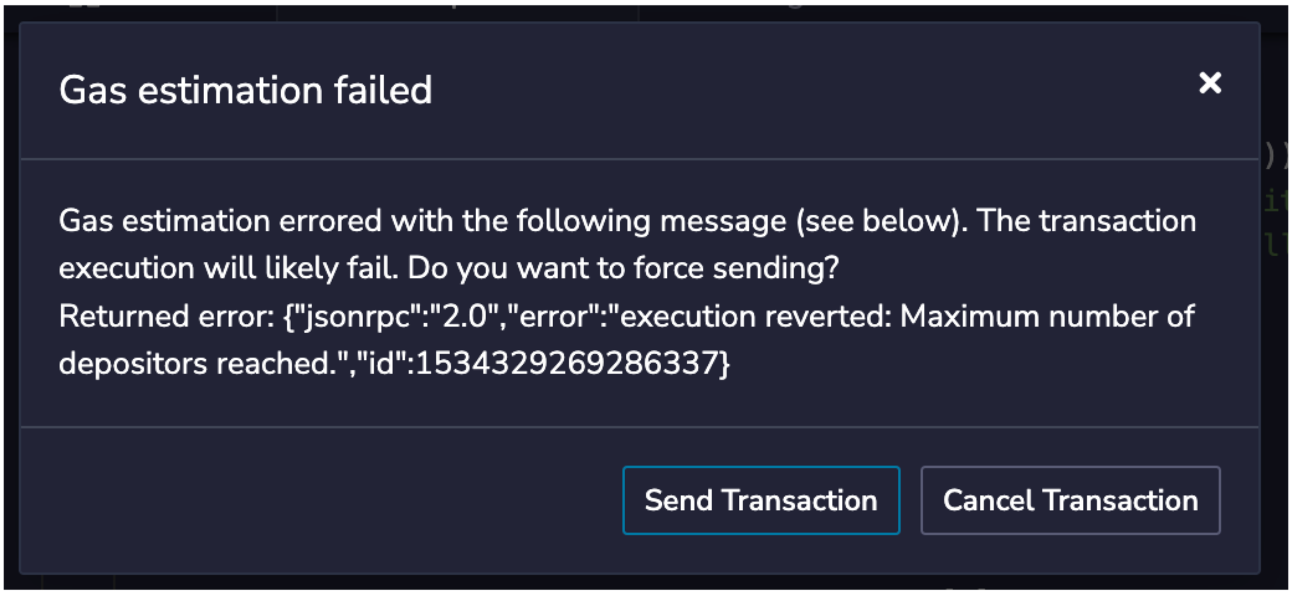
Function: deposit()



```
        // update deposits mapping and depositQueue array
        deposits[msg.sender] = deposits[msg.sender].add(netAmount);
        depositQueue.push(Deposit(msg.sender, netAmount, block.timestamp)); // Add deposit to the queue
        totalDeposits = totalDeposits.add(amount); // Update total deposits
        // Check conditions for automatic fund transfer to withdrawal wallet
        uint256 backupWalletBalance = usdc.balanceOf(backupWallet);
        if (
            depositQueue.length >= 3 &&
            backupWalletBalance >= depositAmount.mul(3) &&
            !paused()
        ) {
            // Send funds to withdrawal wallet
            Deposit memory firstDeposit = depositQueue[0];
            usdc.transfer(withdrawalWallet, firstDeposit.amount.mul(3));
            withdrawalWalletBalance = withdrawalWalletBalance.add(
                firstDeposit.amount.mul(3)
            );
            eligibleWithdrawals[firstDeposit.depositor] = eligibleWithdrawals[
                firstDeposit.depositor
            ].add(firstDeposit.amount.mul(3));
            removeFirstDepositorFromQueue();

        emit Deposited(msg.sender, netAmount, block.timestamp);
    }
```

default_workspace/contracts/SecureDeposit/v3/SecureDeposit.sol
bleWithdrawals - call

Call error:



Gas estimation failed

Gas estimation errored with the following message (see below). The transaction execution will likely fail. Do you want to force sending?
Returned error: {"jsonrpc":"2.0","error":"execution reverted: Maximum number of depositors reached.","id":1534329269286337}

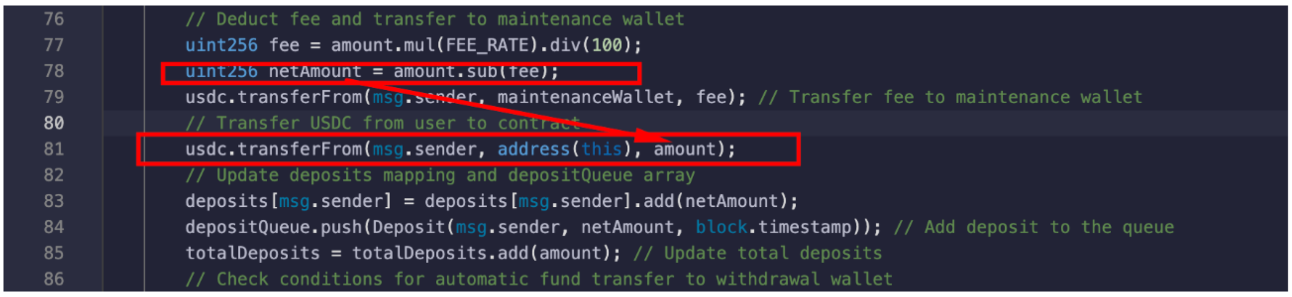
1. Set the deposit amount to 1 USDC.
2. Keep the backup wallet balance 0.
3. By keeping backup wallet balance zero it will never enter the if condition(auto fund transfer to withdrawal wallet) in the deposit function.
4. Keep on depositing until it reaches maxDepositors.
5. Once maxDepositors is reached Try to deposit again.
6. Not able to deposit further and hence

7. No withdrawal logic is called and the contract will be freezed for 2 weeks and no profits are returned to depositors.
8. To unfreeze the contract the admin has to call the revert state function.

Status: This is fixed in the revised smart contract code.

(11) Invalid amount of USDC token transfer to the contract:

Function: Deposit()



```
        // Deduct fee and transfer to maintenance wallet
        uint256 fee = amount.mul(FEE_RATE).div(100);
        uint256 netAmount = amount.sub(fee);
        usdc.transferFrom(msg.sender, maintenanceWallet, fee); // Transfer fee to maintenance wallet
        // Transfer USDC from user to contract
        usdc.transferFrom(msg.sender, address(this), amount);
        // Update deposits mapping and depositQueue array
        deposits[msg.sender] = deposits[msg.sender].add(netAmount);
        depositQueue.push(Deposit(msg.sender, netAmount, block.timestamp)); // Add deposit to the queue
        totalDeposits = totalDeposits.add(amount); // Update total deposits
        // Check conditions for automatic fund transfer to withdrawal wallet
```

After deducting Fee from total deposit amount Transferring total deposit amount of USDC to the contract instead of net amount.

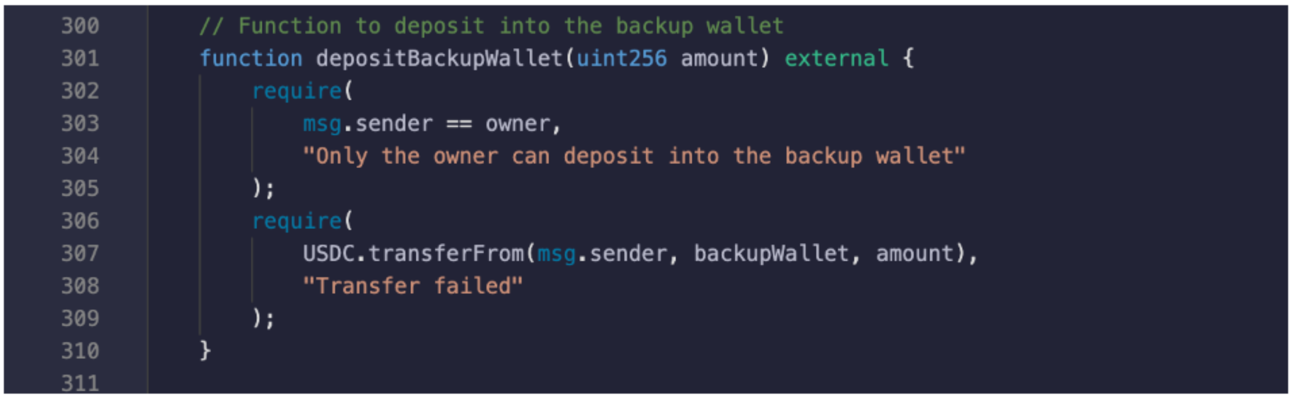
Resolution: We suggest changing the logic and transferring the net amount after deducting the fee from the total deposit amount to the contract.

Status: This is fixed in the revised smart contract code.

Medium

(1) The depositBackupWallet function is not useful:

Function: depositBackupWallet()



```
    // Function to deposit into the backup wallet
    function depositBackupWallet(uint256 amount) external {
        require(
            msg.sender == owner,
            "Only the owner can deposit into the backup wallet"
        );
        require(
            USDC.transferFrom(msg.sender, backupWallet, amount),
            "Transfer failed"
        );
    }
```

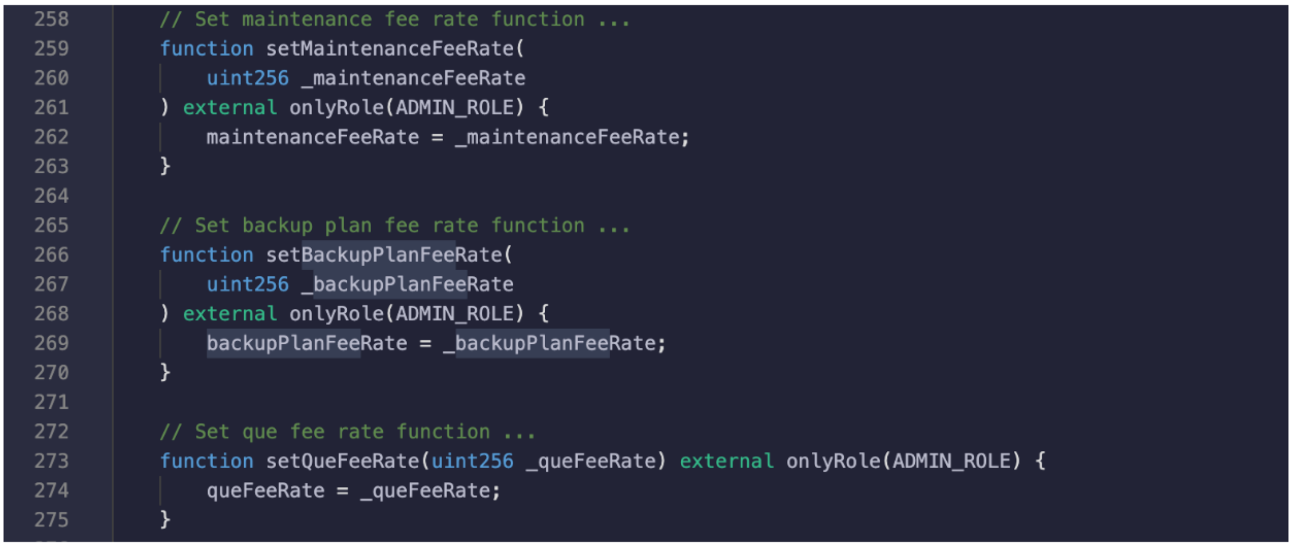
A depositBackupWallet function is used to fund usdc to backupWallet, but backupWallet is nowhere used.

Status: This is fixed in the revised smart contract code.

Low

(1) There is no option to update the fees:

Functions: setMaintenanceFeeRate(), setBackupPlanFeeRate(), setQueFeeRate()



```
    // Set maintenance fee rate function ...
    function setMaintenanceFeeRate(
        uint256 _maintenanceFeeRate
    ) external onlyRole(ADMIN_ROLE) {
        maintenanceFeeRate = _maintenanceFeeRate;
    }

    // Set backup plan fee rate function ...
    function setBackupPlanFeeRate(
        uint256 _backupPlanFeeRate
    ) external onlyRole(ADMIN_ROLE) {
        backupPlanFeeRate = _backupPlanFeeRate;
    }

    // Set que fee rate function ...
    function setQueFeeRate(uint256 _queFeeRate) external onlyRole(ADMIN_ROLE) {
        queFeeRate = _queFeeRate;
    }
```

As per the document, an option should be available to update the fees.

3. Fees & Rates: Verify that the fee calculations in the deposit and withdrawal functions work as intended. Make sure that the fee rates are set properly and that the fees are distributed to the correct wallets. Check if the function to change fee rates works correctly and only accessible to the right role.

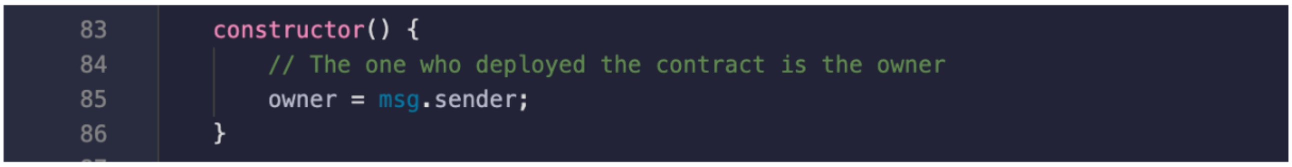
But the update fees functions are using the wrong variables to update the fees, and those variables are nowhere used.

Resolution: Suggest using the correct state variables to update the fees. using the correct functions.

Status: This is fixed in the revised smart contract code.

(2) Use OwnableUpgradeable contract to provide a basic access control mechanism:

Function: Constructor()



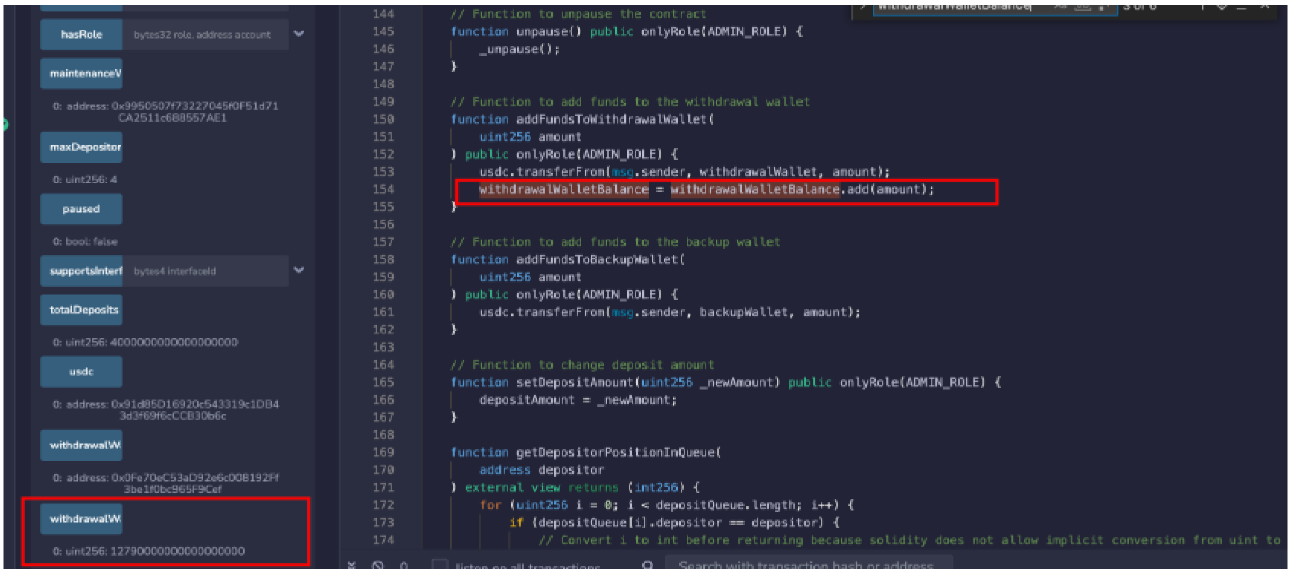
```
constructor() {
    // The one who deployed the contract is the owner
    owner = msg.sender;
}
```

Instead of using state variables to check the owner, use the OwnableUpgradeable contract to provide a basic access control mechanism.

Status: This is fixed in the revised smart contract code.

(3) A WithdrawalWalletBalance is not maintained correctly:

Function: addFundsToWithdrawalWallet()



```
hasRole          bytes32 role, address account
maintenanceV
0: address: 0x9950507f73227045f0F51d71CA2511c688557AE1
maxDepositor
0: uint256: 4
paused
0: bool: false
supportsInterf   bytes4 interfaceId
totalDeposits
0: uint256: 4000000000000000000
usdc
0: address: 0x91d85D16920c543319c1DB43d3f69f6cCCB30b6c
withdrawalW
0: address: 0x0Fe70eC53aD92e6c008192Ff3be1f0bc965F9Cef
withdrawalW
0: uint256: 12790000000000000000
```

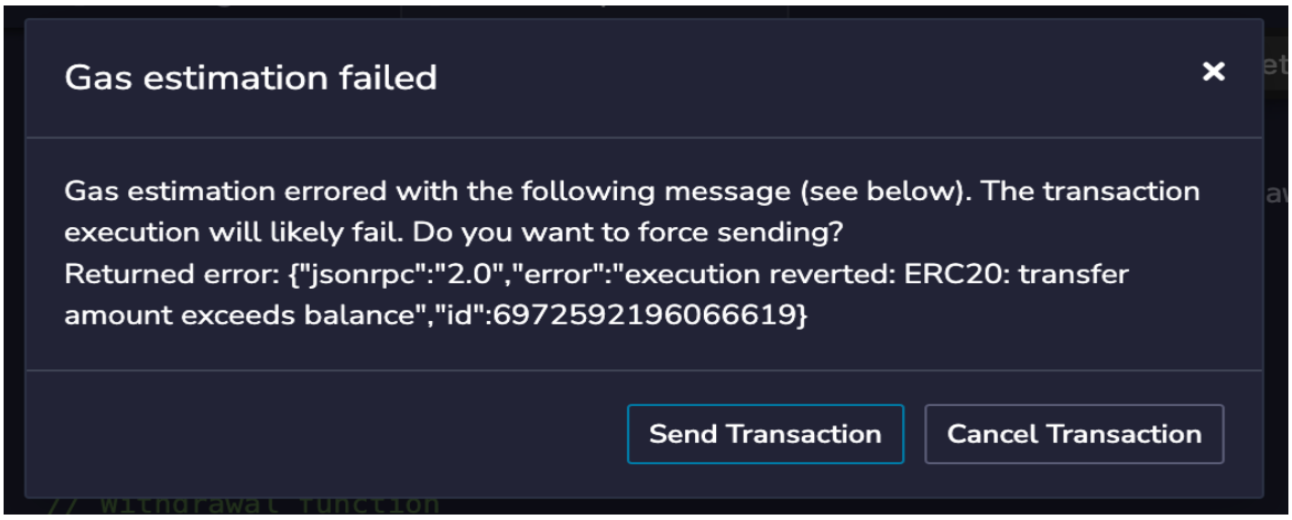
```
// Function to unpause the contract
function unpause() public onlyRole(ADMIN_ROLE) {
    _unpause();
}

// Function to add funds to the withdrawal wallet
function addFundsToWithdrawalWallet(
    uint256 amount
) public onlyRole(ADMIN_ROLE) {
    usdc.transferFrom(msg.sender, withdrawalWallet, amount);
    withdrawalWalletBalance = withdrawalWalletBalance.add(amount);
}

// Function to add funds to the backup wallet
function addFundsToBackupWallet(
    uint256 amount
) public onlyRole(ADMIN_ROLE) {
    usdc.transferFrom(msg.sender, backupWallet, amount);
}

// Function to change deposit amount
function setDepositAmount(uint256 _newAmount) public onlyRole(ADMIN_ROLE) {
    depositAmount = _newAmount;
}

function getDepositorPositionInQueue(
    address depositor
) external view returns (int256) {
    for (uint256 i = 0; i < depositQueue.length; i++) {
        if (depositQueue[i].depositor == depositor) {
            // Convert i to int before returning because solidity does not allow implicit conversion from uint to
```

WithdrawalWalletBalance is updated after adding funds to withdraw wallet which is not correct because tokens can be withdrawn by owner outside the contract since it's a metamask wallet whose control is with the owner of the wallet and not the contract.

Status: This is fixed in the revised smart contract code.

(3) Not able to withdraw USDC after 4th deposit until withdrawal wallet manually funds by the admin:

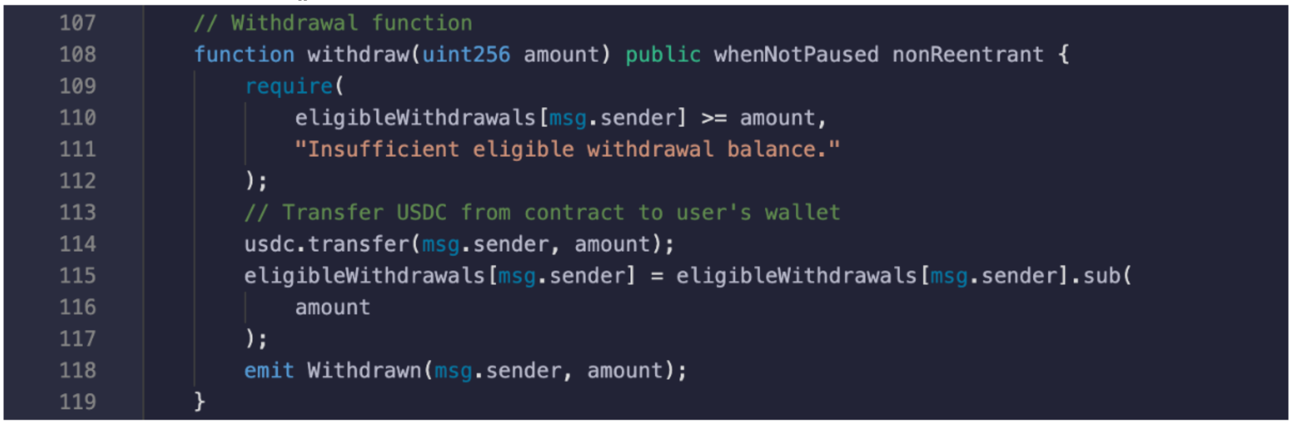
Error:



Gas estimation failed

Gas estimation errored with the following message (see below). The transaction execution will likely fail. Do you want to force sending?
Returned error: {"jsonrpc":"2.0","error":"execution reverted: ERC20: transfer amount exceeds balance","id":6972592196066619}

Function: withdraw()



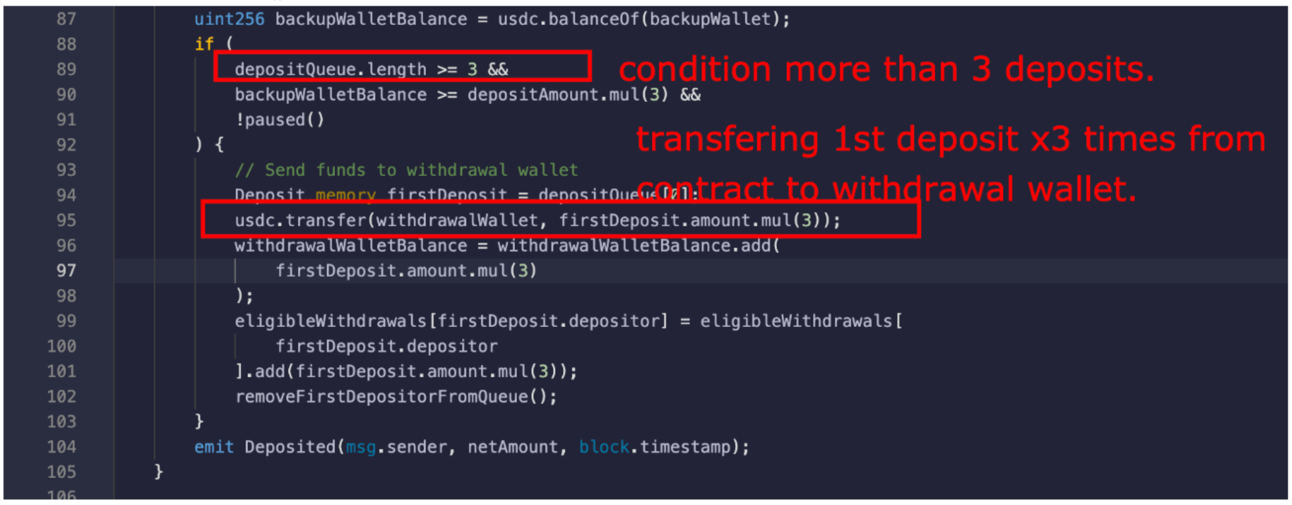
```
// Withdrawal function
function withdraw(uint256 amount) public whenNotPaused nonReentrant {
    require(
        eligibleWithdrawals[msg.sender] >= amount,
        "Insufficient eligible withdrawal balance."
    );
    // Transfer USDC from contract to user's wallet
    usdc.transfer(msg.sender, amount);
    eligibleWithdrawals[msg.sender] = eligibleWithdrawals[msg.sender].sub(
        amount
    );
    emit Withdrawn(msg.sender, amount);
}
```

1. Set the deposit amount to 1 usdc.
2. Deposit 4 times with1 usdc.
3. Try to deposit 5 times.
4. not able to deposit more than 4 times
5.Try to call withdraw function not able to withdraw

<u>The reason is:</u>

1. On 4th deposit the logic is: From 1st 3 deposits(1 usdc in each deposit) after deducting fee(7%) netamount = 0.93 x 3= 2.79 usdc is accumulated in the contract.
2. For 4th deposit it will enter if condition(refer below code snippet) its withdrawing 1st deposited amount which is 0.93 x3 = 2.79 usdc from contract to withdrawal wallet.
3. So when user calls withdraw function no sufficient balance in contract for withdrawal not able to withdraw.

Function: deposit()


condition more than 3 deposits.
transfering 1st deposit x3 times from contract to withdrawal wallet.

```
        uint256 backupWalletBalance = usdc.balanceOf(backupWallet);
        if (
            depositQueue.length >= 3 &&
            backupWalletBalance >= depositAmount.mul(3) &&
            !paused()
        ) {
            // Send funds to withdrawal wallet
            Deposit memory firstDeposit = depositQueue[0];
            usdc.transfer(withdrawalWallet, firstDeposit.amount.mul(3));
            withdrawalWalletBalance = withdrawalWalletBalance.add(
                firstDeposit.amount.mul(3)
            );
            eligibleWithdrawals[firstDeposit.depositor] = eligibleWithdrawals[
                firstDeposit.depositor
            ].add(firstDeposit.amount.mul(3));
            removeFirstDepositorFromQueue();
        }
        emit Deposited(msg.sender, netAmount, block.timestamp);
    }
```

Status: **Open** (until withdrawal wallet manually funded by the admin)

Very Low / Informational / Best practices:

(1) Unused variables:

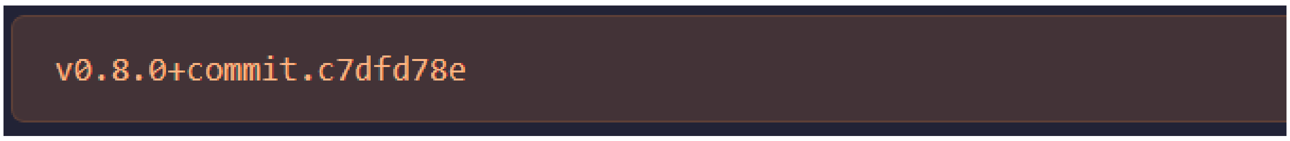
There are variables defined but not used anywhere.

- totalQueuedUsers
- depositCounter
- withdrawalCounter
- withdrawalQueue
- taxWallet
- usdcPrice

Resolution: Remove unused variables from the code.

Status: This is fixed in the revised smart contract code.

(2) Please use the latest compiler version when deploying the contract:



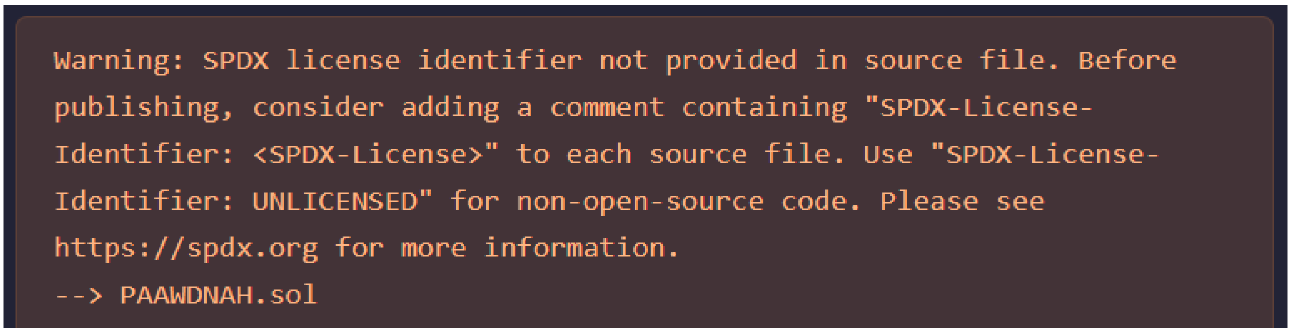
```
v0.8.0+commit.c7dfd78e
```

This is not a severe issue, but we suggest using the latest compiler version at the time of contract deployment, which is 0.8.19 at the time of this audit. Using the latest compiler version is always recommended, which prevents any compiler level issues.

Status: This is fixed in the revised smart contract code.

(3) Warning: SPDX licence identifier:

Warning Error:



```
Warning: SPDX license identifier not provided in source file. Before
publishing, consider adding a comment containing "SPDX-License-
Identifier: <SPDX-License>" to each source file. Use "SPDX-License-
Identifier: UNLICENSED" for non-open-source code. Please see
https://spdx.org for more information.
--> PAAWDNAH.sol
```

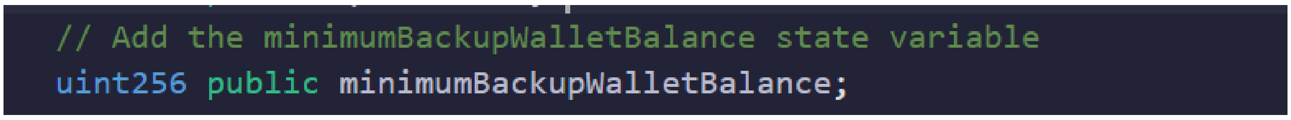
Warning: SPDX license identifier is not provided in the source file.

Resolution: We suggest adding SPDX-License-Identifier.

Status: This is fixed in the revised smart contract code.

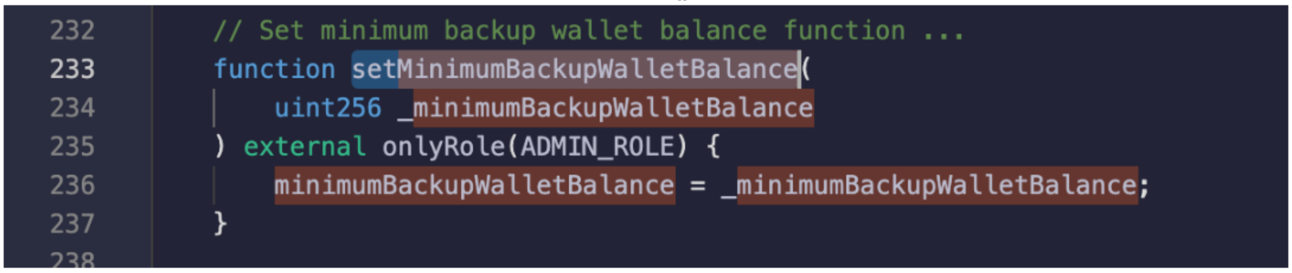
(4) Redundant variable:

Variable: minimumBackupWalletBalance



```
// Add the minimumBackupWalletBalance state variable
uint256 public minimumBackupWalletBalance;
```

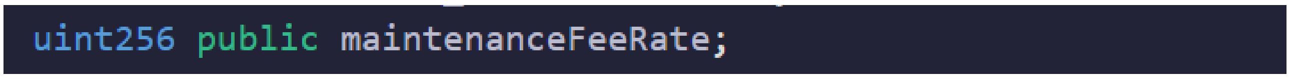
Function: setMinimumBackupWalletBalance()



```
// Set minimum backup wallet balance function ...
function setMinimumBackupWalletBalance(
    uint256 _minimumBackupWalletBalance
) external onlyRole(ADMIN_ROLE) {
    minimumBackupWalletBalance = _minimumBackupWalletBalance;
}
```

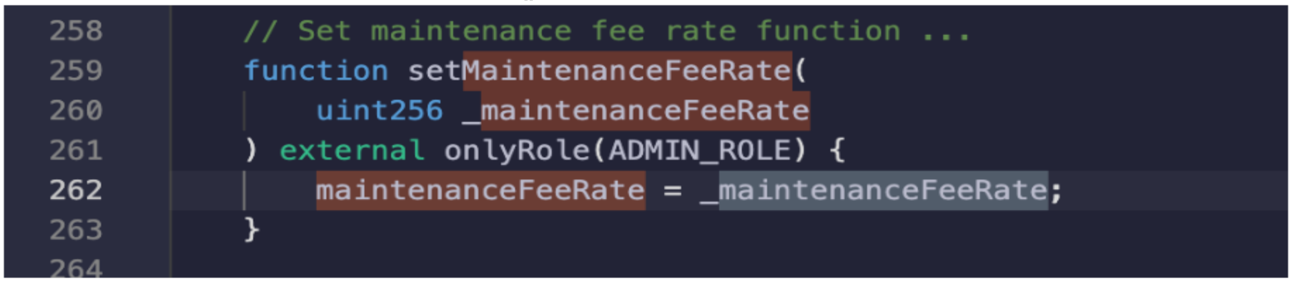
The minimumBackupWalletBalance is defined but nowhere used; it is just used to update the value using the setMinimumBackupWalletBalance function.

Variable: maintenanceFeeRate



```
uint256 public maintenanceFeeRate;
```

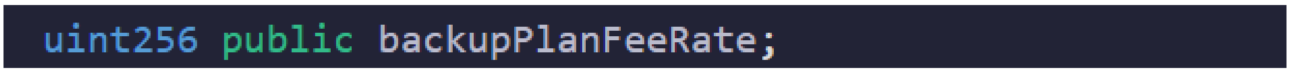
Function: setMaintenanceFeeRate()



```
// Set maintenance fee rate function ...
function setMaintenanceFeeRate(
    uint256 _maintenanceFeeRate
) external onlyRole(ADMIN_ROLE) {
    maintenanceFeeRate = _maintenanceFeeRate;
}
```

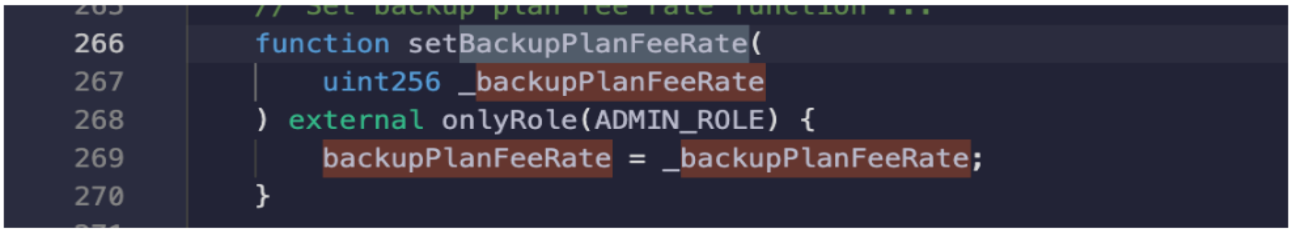
The maintenanceFeeRate is defined but nowhere used; it is just used to update the value using the setMaintenanceFeeRate function.

Variable: backupPlanFeeRate



```
uint256 public backupPlanFeeRate;
```

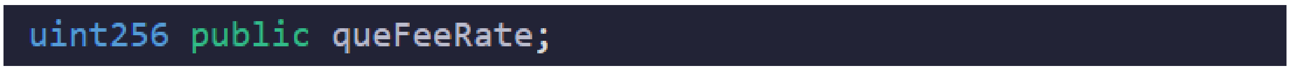
Function: setBackupPlanFeeRate()



```
// Set backup plan fee rate function ...
function setBackupPlanFeeRate(
    uint256 _backupPlanFeeRate
) external onlyRole(ADMIN_ROLE) {
    backupPlanFeeRate = _backupPlanFeeRate;
}
```

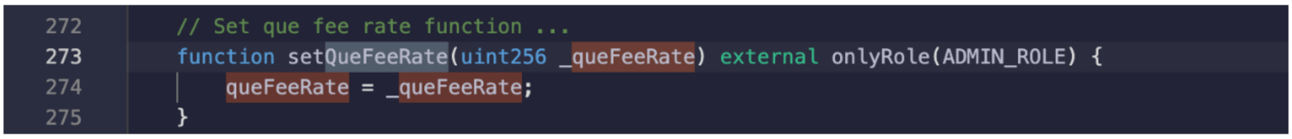
The backupPlanFeeRate is defined but nowhere used; it is just used to update the value using the setBackupPlanFeeRate function.

Variable: queFeeRate



```
uint256 public queFeeRate;
```

Function: setQueFeeRate()



```
// Set que fee rate function ...
function setQueFeeRate(uint256 _queFeeRate) external onlyRole(ADMIN_ROLE) {
    queFeeRate = _queFeeRate;
}
```

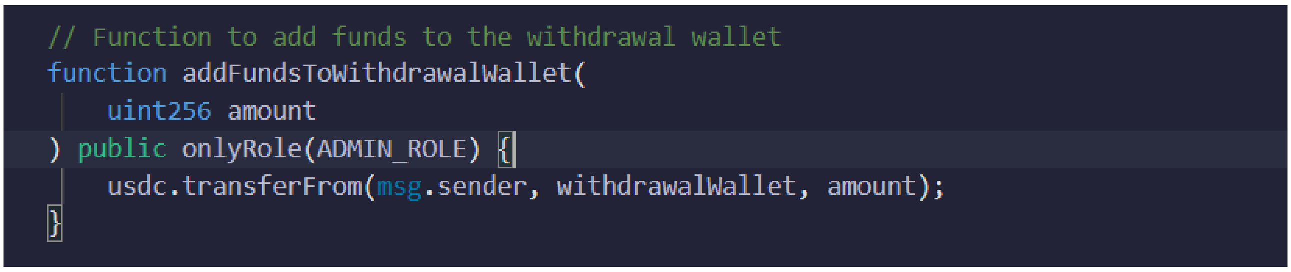
The queFeeRate is defined but nowhere used; it is just used to update the value using the setQueFeeRate function.

Resolution: If the variable is not used, please suggest removing it.

Status: This is fixed in the revised smart contract code.

(5) Redundant function:

Function: addFundsToWithdrawalWallet()



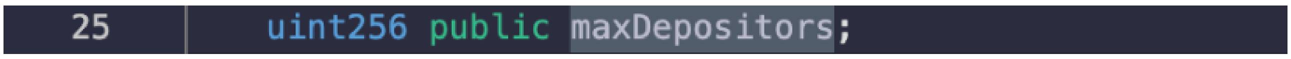
```
// Function to add funds to the withdrawal wallet
function addFundsToWithdrawalWallet(
    uint256 amount
) public onlyRole(ADMIN_ROLE) {
    usdc.transferFrom(msg.sender, withdrawalWallet, amount);
}
```

The new code introduced requires a statement checking the balance of the sender before transfer which is not needed because if no funds the transaction will not trigger it will auto reject saying insufficient funds.

Status: This is fixed in the revised smart contract code.

(6) Unused variables:

Variable: maxDepositors



```
uint256 public maxDepositors;
```

Variable is not used in contract.

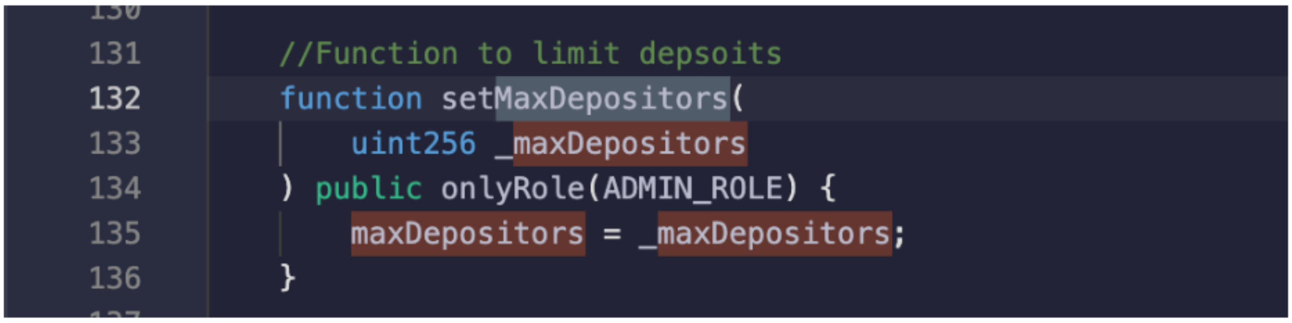
- maxDepositors

Resolution: Remove unused variable.

Status: This is fixed in the revised smart contract code.

(7) Function to update unused variable:

Function: setMaxDepositors()



```
    //Function to limit depsoits
    function setMaxDepositors(
        uint256 _maxDepositors
    ) public onlyRole(ADMIN_ROLE) {
        maxDepositors = _maxDepositors;
    }
```

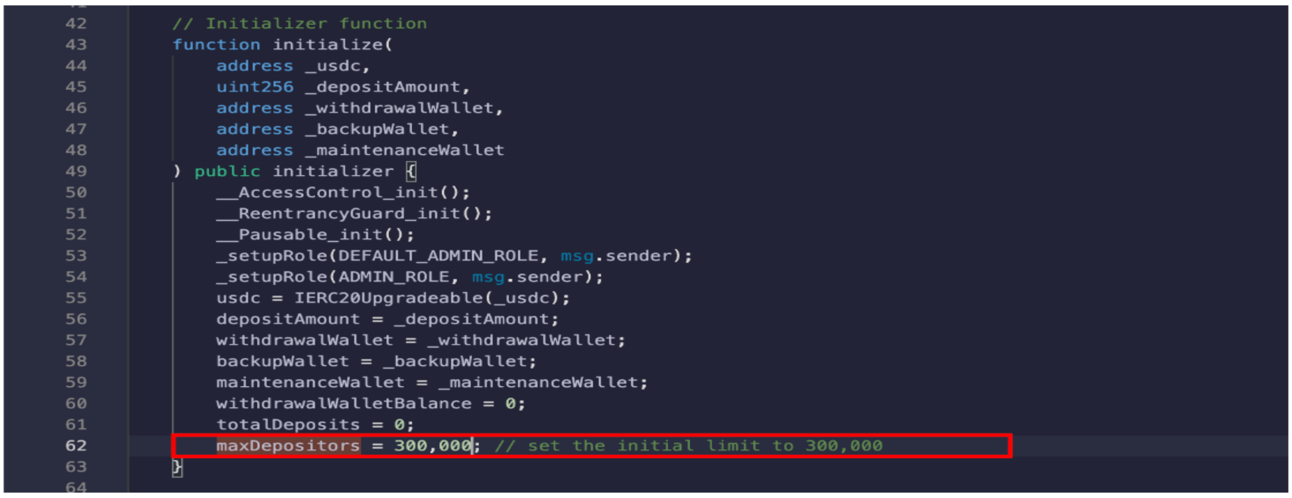
Function to update the variable which is not used in contract.

Resolution: Remove function which is not useful.

Status: This is fixed in the revised smart contract code.

(8) Updating unused variable while initializing the contract:

Function: initialize()



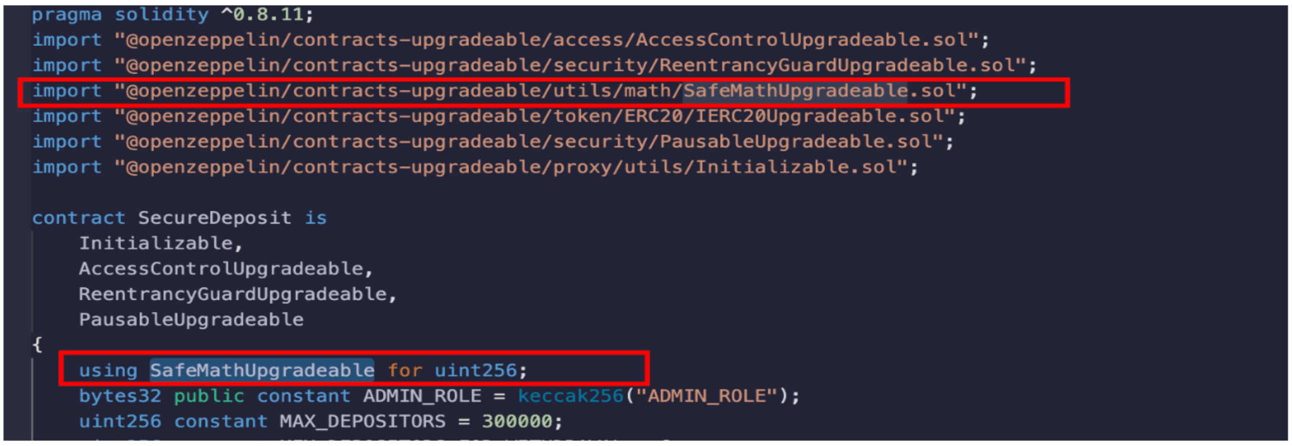
```
// Initializer function
function initialize(
    address _usdc,
    uint256 _depositAmount,
    address _withdrawalWallet,
    address _backupWallet,
    address _maintenanceWallet
) public initializer {
    __AccessControl_init();
    __ReentrancyGuard_init();
    __Pausable_init();
    _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _setupRole(ADMIN_ROLE, msg.sender);
    usdc = IERC20Upgradeable(_usdc);
    depositAmount = _depositAmount;
    withdrawalWallet = _withdrawalWallet;
    backupWallet = _backupWallet;
    maintenanceWallet = _maintenanceWallet;
    withdrawalWalletBalance = 0;
    totalDeposits = 0;
    maxDepositors = 300,000; // set the initial limit to 300,000
}
```

Updating unused variables while initializing the contract.

Resolution: Remove mentioned line of code.

Status: This is fixed in the revised smart contract code.

(9) SafeMath Library:



```
pragma solidity ^0.8.11;
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol";
import "@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";

contract SecureDeposit is
    Initializable,
    AccessControlUpgradeable,
    ReentrancyGuardUpgradeable,
    PausableUpgradeable
{
    using SafeMathUpgradeable for uint256;
    bytes32 public constant ADMIN_ROLE = keccak256("ADMIN_ROLE");
    uint256 constant MAX_DEPOSITORS = 300000;
```

The SafeMath Library is used in this contract code, but if the compiler version is greater than or equal to 0.8.0, it will not be required to be used because Solidity automatically handles overflow and underflow.

Resolution: Remove the SafeMath library and use normal math operators. It will improve code size and reduce gas consumption.

Status: Acknowledged, as the impact of this is not significant.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

AccessControlUpgradeable.sol

- grantRole: Grant role and address can be assigned by the admin.
- revokeRole: Revoke role and address can be assigned by the admin.
- renounceRole: Renounce roles for self by the admin.

PAAWDNAH.sol

- pause: Triggers stopped state can be set by the admin.
- unpause: Returns to normal state can be set by the admin.
- setMaxDepositors: Maximum depositors values can be set by the admin.
- addFundsToWithdrawalWallet: Add funds to the withdrawal wallet by the admin.
- addFundsToBackupWallet: Add funds to the backup wallet by the admin.
- setDepositAmount: Update Deposit amount by the admin.
- revertState: Revert state can be set by the admin.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of a file and we have used all possible tests based on given objects as files. We have observed 11 high severity issues, 1 medium severity issue, 4 low severity issues and 9 very low severity issues in the smart contract. We confirm that 11 high severity issues, 1 medium severity issue, 3 low severity issues, 8 informational severity issues are fixed in the revised smart contract code. **So, the smart contracts are ready for the mainnet deployment.**

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

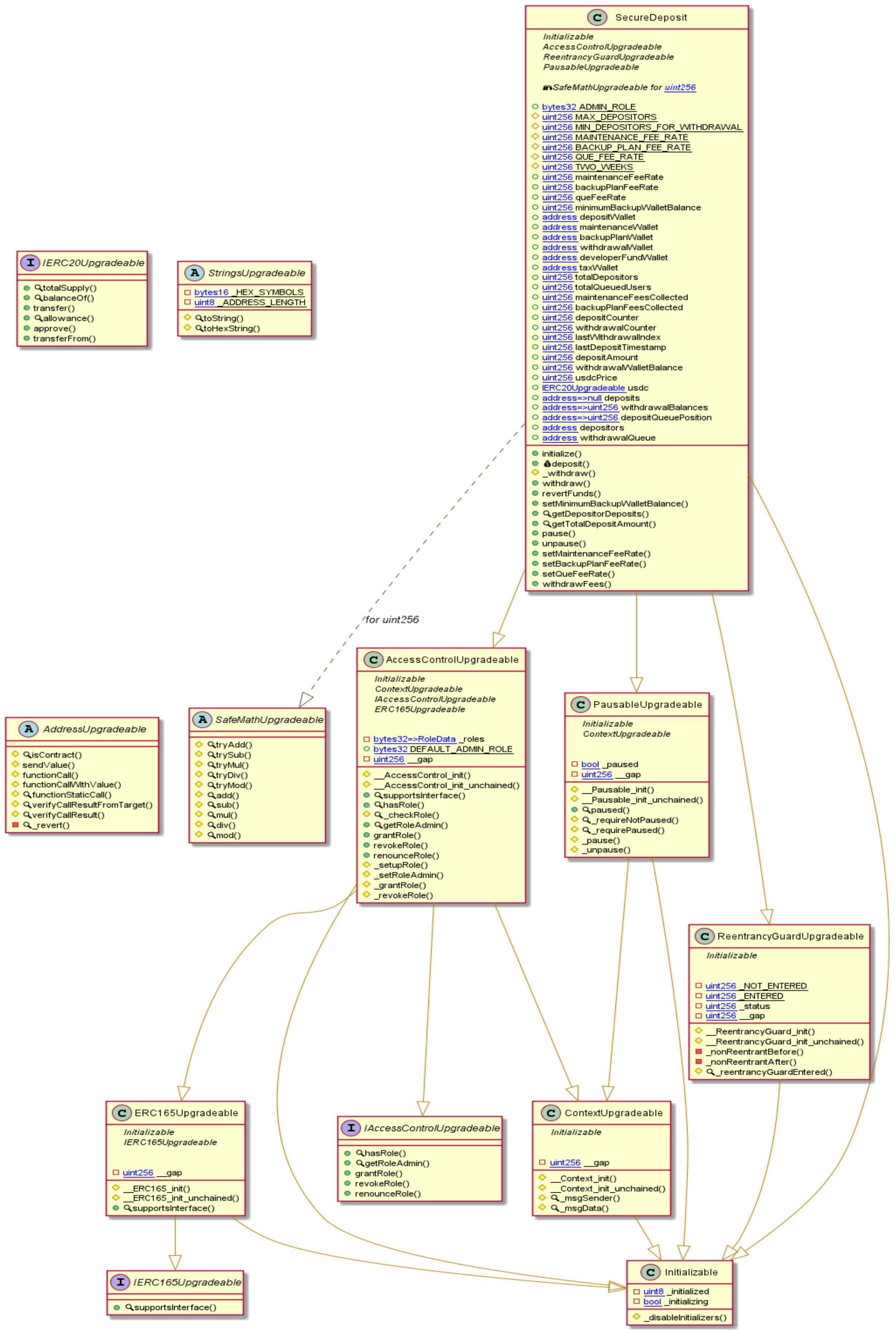
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

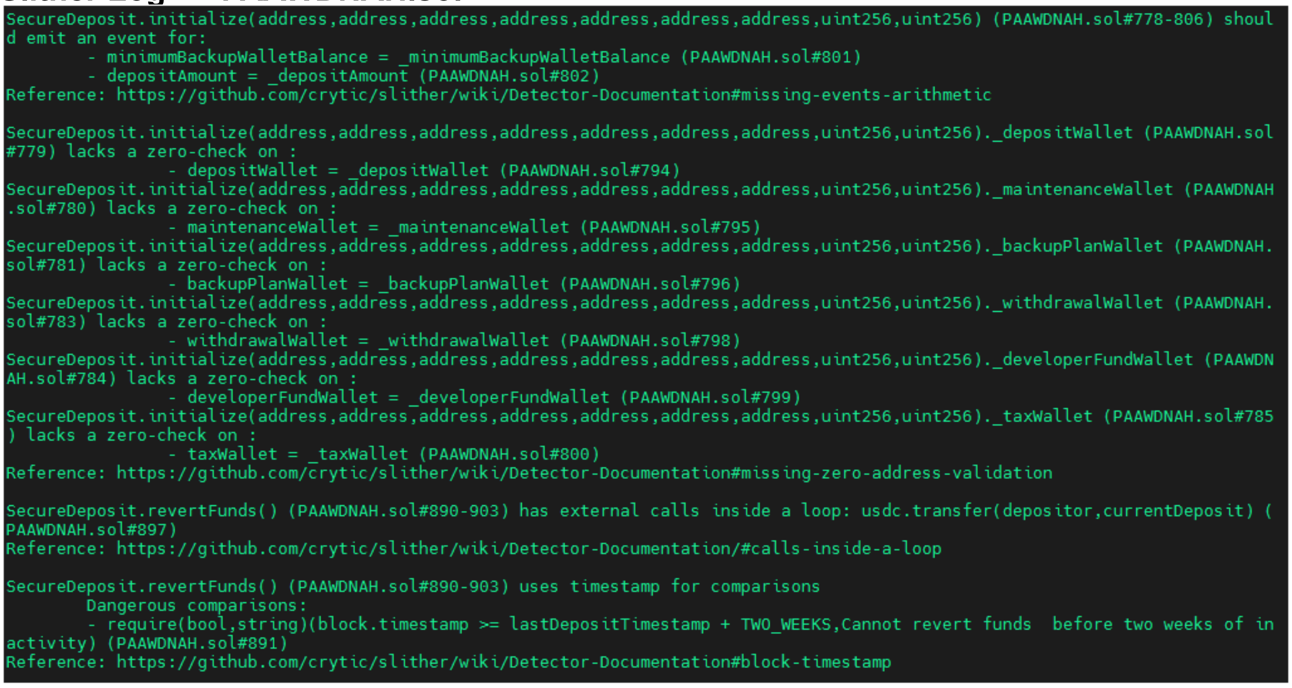
Appendix

Code Flow Diagram - PAAWDNAH Token

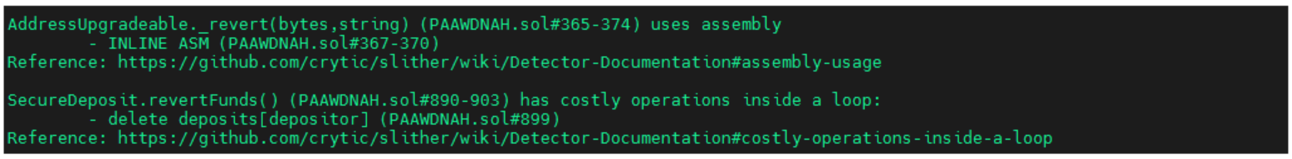


Slither Results Log

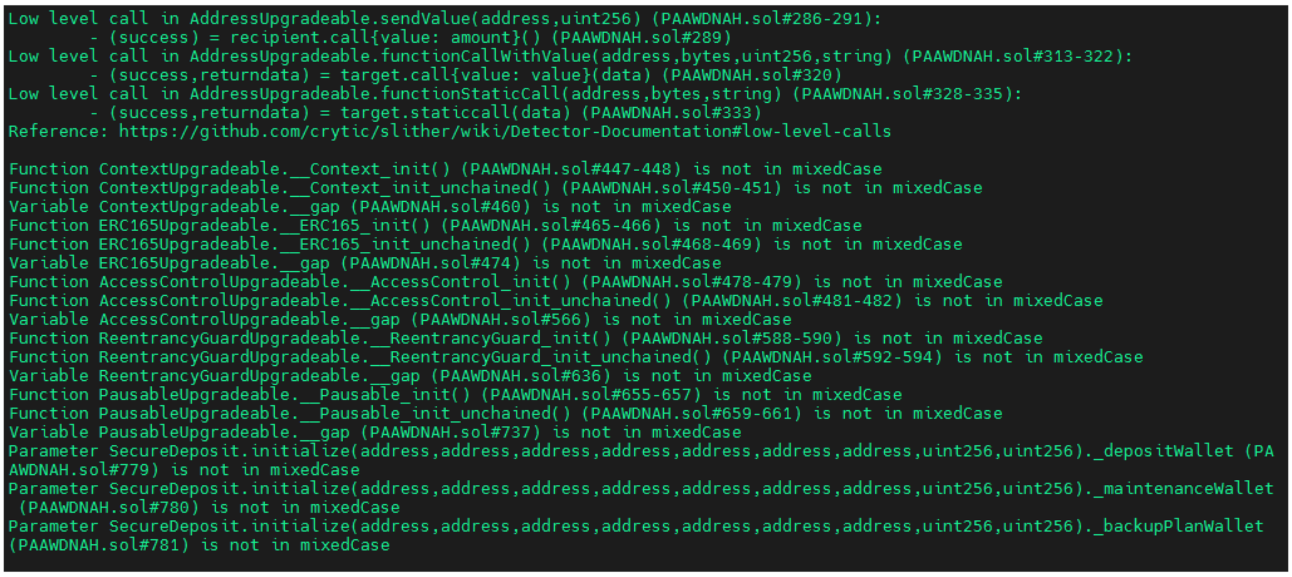
Slither Log >> PAAWDNAH.sol



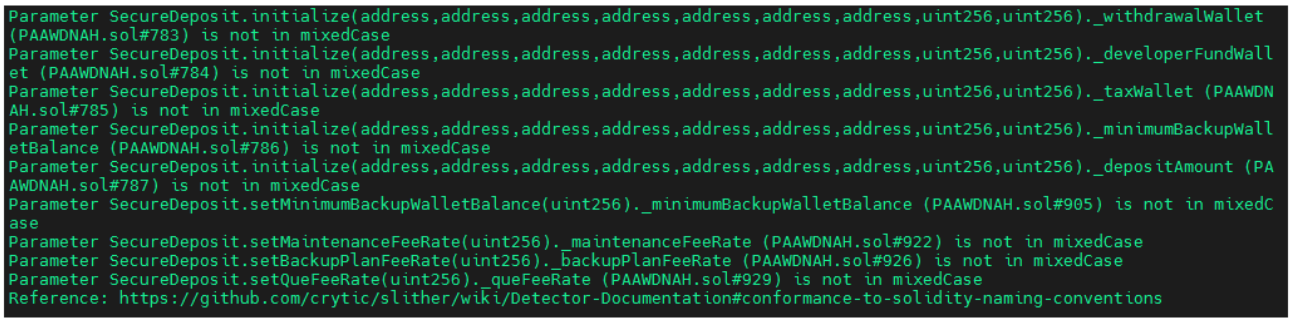
```
SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256) (PAAWDNAH.sol#778-806) shoul
d emit an event for:
        - minimumBackupWalletBalance = _minimumBackupWalletBalance (PAAWDNAH.sol#801)
        - depositAmount = _depositAmount (PAAWDNAH.sol#802)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256)._depositWallet (PAAWDNAH.sol
#779) lacks a zero-check on :
                - depositWallet = _depositWallet (PAAWDNAH.sol#794)
SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256)._maintenanceWallet (PAAWDNAH
.sol#780) lacks a zero-check on :
                - maintenanceWallet = _maintenanceWallet (PAAWDNAH.sol#795)
SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256)._backupPlanWallet (PAAWDNAH.
sol#781) lacks a zero-check on :
                - backupPlanWallet = _backupPlanWallet (PAAWDNAH.sol#796)
SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256)._withdrawalWallet (PAAWDNAH.
sol#783) lacks a zero-check on :
                - withdrawalWallet = _withdrawalWallet (PAAWDNAH.sol#798)
SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256)._developerFundWallet (PAAWDN
AH.sol#784) lacks a zero-check on :
                - developerFundWallet = _developerFundWallet (PAAWDNAH.sol#799)
SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256)._taxWallet (PAAWDNAH.sol#785
) lacks a zero-check on :
                - taxWallet = _taxWallet (PAAWDNAH.sol#800)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

SecureDeposit.revertFunds() (PAAWDNAH.sol#890-903) has external calls inside a loop: usdc.transfer(depositor,currentDeposit) (
PAAWDNAH.sol#897)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

SecureDeposit.revertFunds() (PAAWDNAH.sol#890-903) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= lastDepositTimestamp + TWO_WEEKS,Cannot revert funds  before two weeks of in
activity) (PAAWDNAH.sol#891)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```



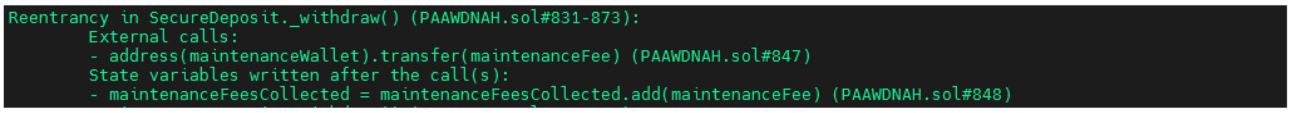
```
AddressUpgradeable._revert(bytes,string) (PAAWDNAH.sol#365-374) uses assembly
        - INLINE ASM (PAAWDNAH.sol#367-370)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

SecureDeposit.revertFunds() (PAAWDNAH.sol#890-903) has costly operations inside a loop:
        - delete deposits[depositor] (PAAWDNAH.sol#899)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
```



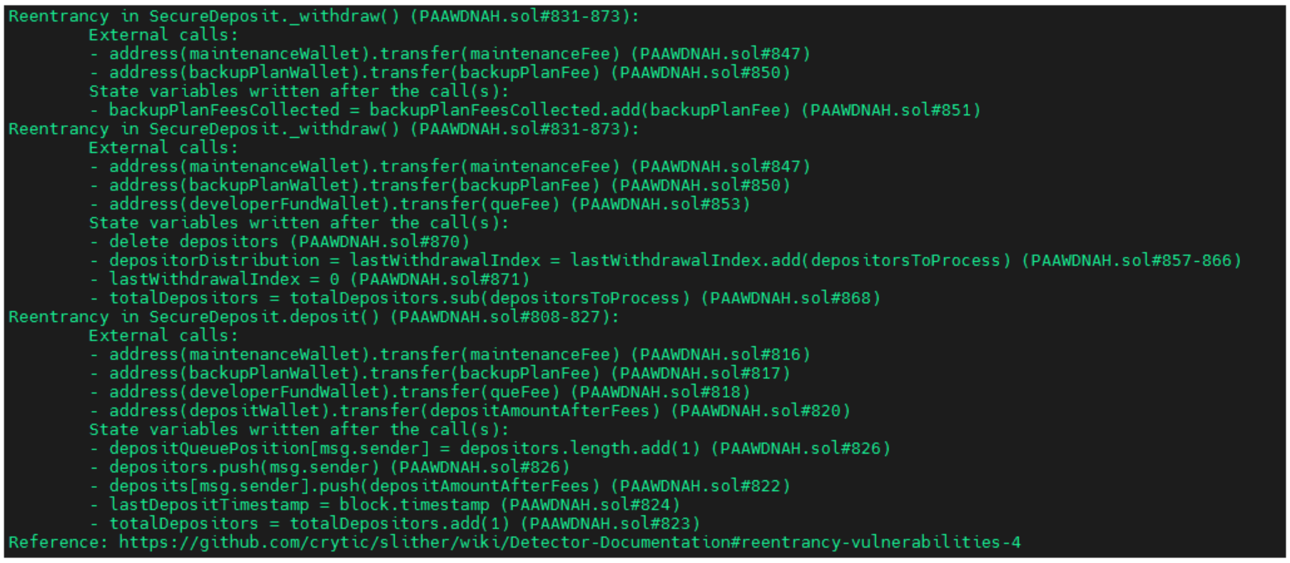
```
Low level call in AddressUpgradeable.sendValue(address,uint256) (PAAWDNAH.sol#286-291):
        - (success) = recipient.call{value: amount}() (PAAWDNAH.sol#289)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (PAAWDNAH.sol#313-322):
        - (success,returndata) = target.call{value: value}(data) (PAAWDNAH.sol#320)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (PAAWDNAH.sol#328-335):
        - (success,returndata) = target.staticcall(data) (PAAWDNAH.sol#333)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function ContextUpgradeable.__Context_init() (PAAWDNAH.sol#447-448) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (PAAWDNAH.sol#450-451) is not in mixedCase
Variable ContextUpgradeable.__gap (PAAWDNAH.sol#460) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init() (PAAWDNAH.sol#465-466) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init_unchained() (PAAWDNAH.sol#468-469) is not in mixedCase
Variable ERC165Upgradeable.__gap (PAAWDNAH.sol#474) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init() (PAAWDNAH.sol#478-479) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init_unchained() (PAAWDNAH.sol#481-482) is not in mixedCase
Variable AccessControlUpgradeable.__gap (PAAWDNAH.sol#566) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init() (PAAWDNAH.sol#588-590) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained() (PAAWDNAH.sol#592-594) is not in mixedCase
Variable ReentrancyGuardUpgradeable.__gap (PAAWDNAH.sol#636) is not in mixedCase
Function PausableUpgradeable.__Pausable_init() (PAAWDNAH.sol#655-657) is not in mixedCase
Function PausableUpgradeable.__Pausable_init_unchained() (PAAWDNAH.sol#659-661) is not in mixedCase
Variable PausableUpgradeable.__gap (PAAWDNAH.sol#737) is not in mixedCase
Parameter SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256)._depositWallet (PA
AWDNAH.sol#779) is not in mixedCase
Parameter SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256)._maintenanceWallet
 (PAAWDNAH.sol#780) is not in mixedCase
Parameter SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256)._backupPlanWallet
(PAAWDNAH.sol#781) is not in mixedCase
```



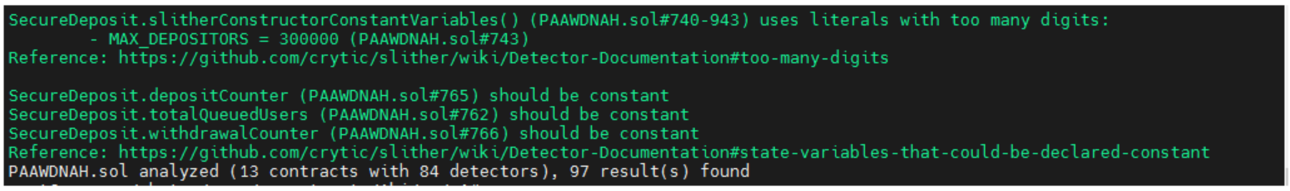
```
Parameter SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256)._withdrawalWallet
(PAAWDNAH.sol#783) is not in mixedCase
Parameter SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256)._developerFundWall
et (PAAWDNAH.sol#784) is not in mixedCase
Parameter SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256)._taxWallet (PAAWDN
AH.sol#785) is not in mixedCase
Parameter SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256)._minimumBackupWall
etBalance (PAAWDNAH.sol#786) is not in mixedCase
Parameter SecureDeposit.initialize(address,address,address,address,address,address,address,uint256,uint256)._depositAmount (PA
AWDNAH.sol#787) is not in mixedCase
Parameter SecureDeposit.setMinimumBackupWalletBalance(uint256)._minimumBackupWalletBalance (PAAWDNAH.sol#905) is not in mixedC
ase
Parameter SecureDeposit.setMaintenanceFeeRate(uint256)._maintenanceFeeRate (PAAWDNAH.sol#922) is not in mixedCase
Parameter SecureDeposit.setBackupPlanFeeRate(uint256)._backupPlanFeeRate (PAAWDNAH.sol#926) is not in mixedCase
Parameter SecureDeposit.setQueFeeRate(uint256)._queFeeRate (PAAWDNAH.sol#929) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```



```
Reentrancy in SecureDeposit._withdraw() (PAAWDNAH.sol#831-873):
        External calls:
        - address(maintenanceWallet).transfer(maintenanceFee) (PAAWDNAH.sol#847)
        State variables written after the call(s):
        - maintenanceFeesCollected = maintenanceFeesCollected.add(maintenanceFee) (PAAWDNAH.sol#848)
```



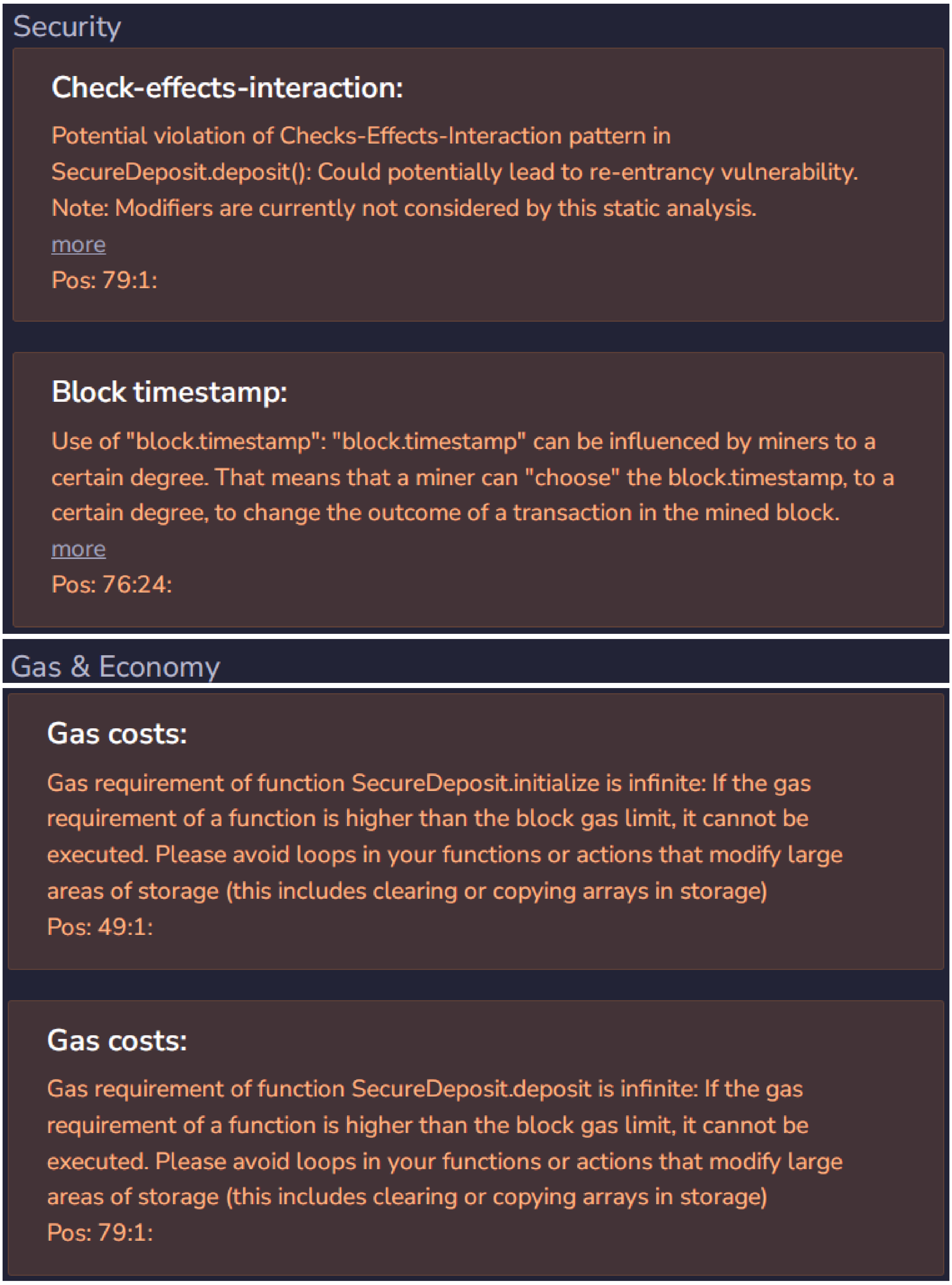
```
Reentrancy in SecureDeposit._withdraw() (PAAWDNAH.sol#831-873):
        External calls:
        - address(maintenanceWallet).transfer(maintenanceFee) (PAAWDNAH.sol#847)
        - address(backupPlanWallet).transfer(backupPlanFee) (PAAWDNAH.sol#850)
        State variables written after the call(s):
        - backupPlanFeesCollected = backupPlanFeesCollected.add(backupPlanFee) (PAAWDNAH.sol#851)
Reentrancy in SecureDeposit._withdraw() (PAAWDNAH.sol#831-873):
        External calls:
        - address(maintenanceWallet).transfer(maintenanceFee) (PAAWDNAH.sol#847)
        - address(backupPlanWallet).transfer(backupPlanFee) (PAAWDNAH.sol#850)
        - address(developerFundWallet).transfer(queFee) (PAAWDNAH.sol#853)
        State variables written after the call(s):
        - delete depositors (PAAWDNAH.sol#870)
        - depositorDistribution = lastWithdrawalIndex = lastWithdrawalIndex.add(depositorsToProcess) (PAAWDNAH.sol#857-866)
        - lastWithdrawalIndex = 0 (PAAWDNAH.sol#871)
        - totalDepositors = totalDepositors.sub(depositorsToProcess) (PAAWDNAH.sol#868)
Reentrancy in SecureDeposit.deposit() (PAAWDNAH.sol#808-827):
        External calls:
        - address(maintenanceWallet).transfer(maintenanceFee) (PAAWDNAH.sol#816)
        - address(backupPlanWallet).transfer(backupPlanFee) (PAAWDNAH.sol#817)
        - address(developerFundWallet).transfer(queFee) (PAAWDNAH.sol#818)
        - address(depositWallet).transfer(depositAmountAfterFees) (PAAWDNAH.sol#820)
        State variables written after the call(s):
        - depositQueuePosition[msg.sender] = depositors.length.add(1) (PAAWDNAH.sol#826)
        - depositors.push(msg.sender) (PAAWDNAH.sol#826)
        - deposits[msg.sender].push(depositAmountAfterFees) (PAAWDNAH.sol#822)
        - lastDepositTimestamp = block.timestamp (PAAWDNAH.sol#824)
        - totalDepositors = totalDepositors.add(1) (PAAWDNAH.sol#823)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
```



```
SecureDeposit.slitherConstructorConstantVariables() (PAAWDNAH.sol#740-943) uses literals with too many digits:
        - MAX_DEPOSITORS = 300000 (PAAWDNAH.sol#743)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

SecureDeposit.depositCounter (PAAWDNAH.sol#765) should be constant
SecureDeposit.totalQueuedUsers (PAAWDNAH.sol#762) should be constant
SecureDeposit.withdrawalCounter (PAAWDNAH.sol#766) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
PAAWDNAH.sol analyzed (13 contracts with 84 detectors), 97 result(s) found
```

Solidity Static Analysis

PAAWDNAH.sol



Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in SecureDeposit.deposit(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 79:1:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 76:24:

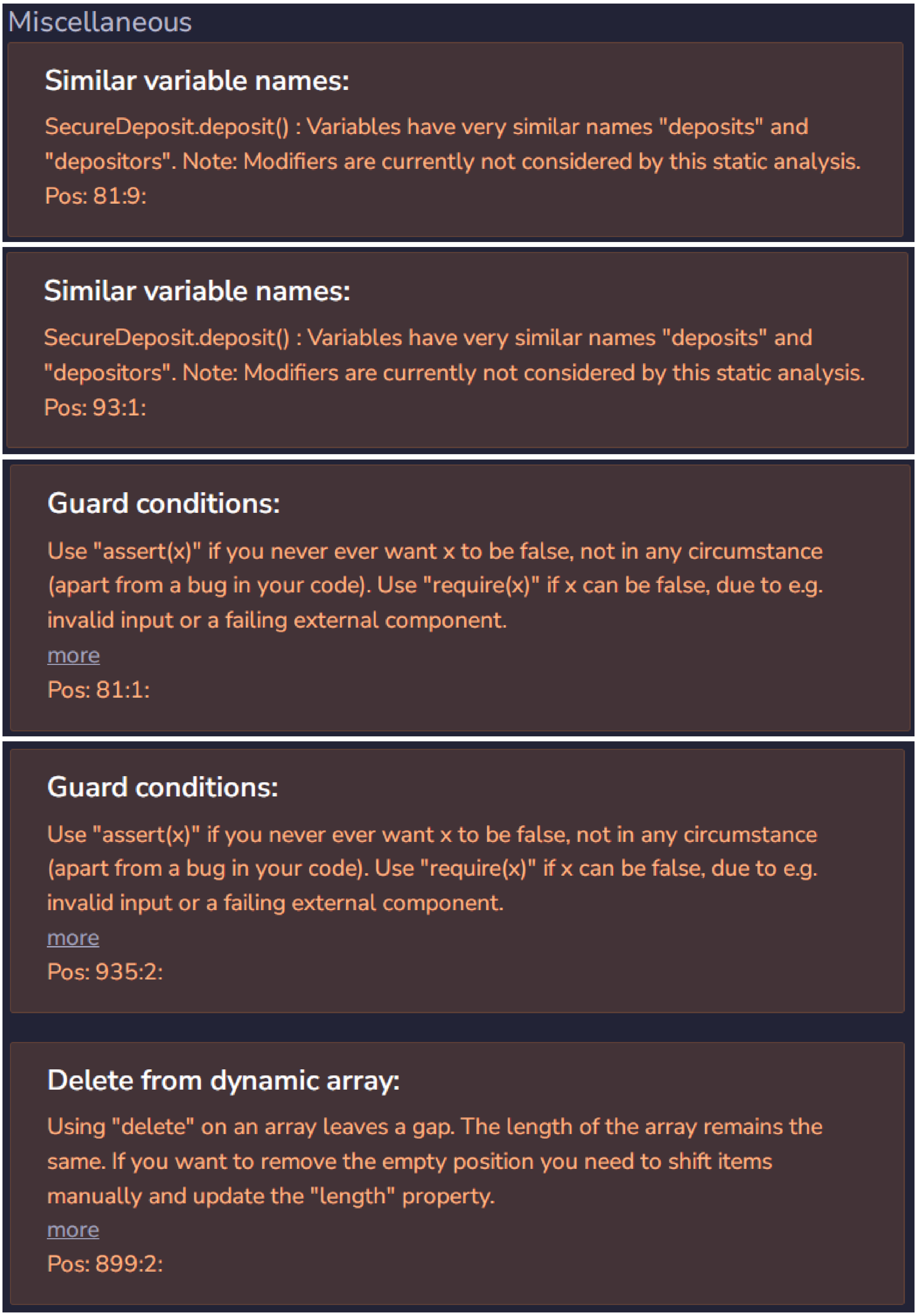
Gas & Economy

Gas costs:

Gas requirement of function SecureDeposit.initialize is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 49:1:

Gas costs:

Gas requirement of function SecureDeposit.deposit is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 79:1:



Miscellaneous

Similar variable names:

SecureDeposit.deposit() : Variables have very similar names "deposits" and "depositors". Note: Modifiers are currently not considered by this static analysis.
Pos: 81:9:

Similar variable names:

SecureDeposit.deposit() : Variables have very similar names "deposits" and "depositors". Note: Modifiers are currently not considered by this static analysis.
Pos: 93:1:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 81:1:

Guard conditions:

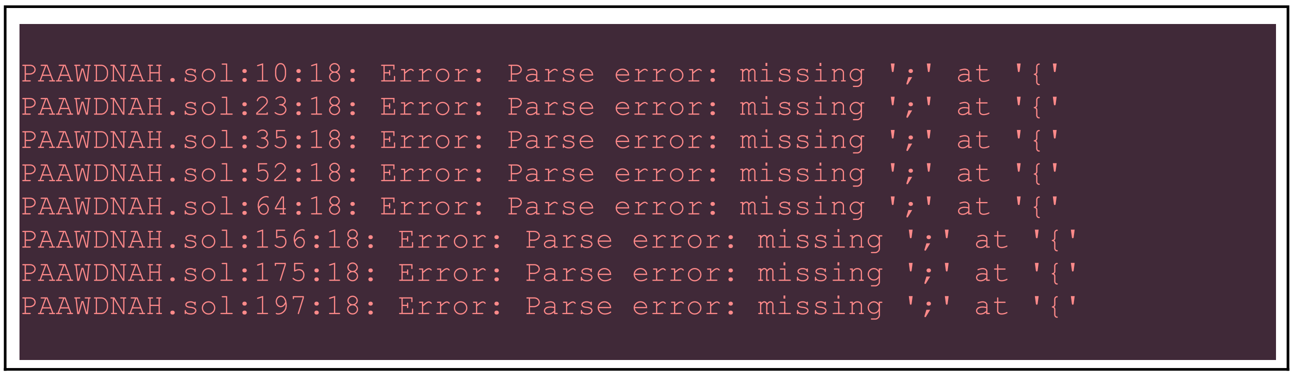
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 935:2:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
more
Pos: 899:2:

Solhint Linter

PAAWDNAH.sol



```
PAAWDNAH.sol:10:18: Error: Parse error: missing ';' at '{'
PAAWDNAH.sol:23:18: Error: Parse error: missing ';' at '{'
PAAWDNAH.sol:35:18: Error: Parse error: missing ';' at '{'
PAAWDNAH.sol:52:18: Error: Parse error: missing ';' at '{'
PAAWDNAH.sol:64:18: Error: Parse error: missing ';' at '{'
PAAWDNAH.sol:156:18: Error: Parse error: missing ';' at '{'
PAAWDNAH.sol:175:18: Error: Parse error: missing ';' at '{'
PAAWDNAH.sol:197:18: Error: Parse error: missing ';' at '{'
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.