QuillAudits

Audit Report, April, 2024

For

Mabrook

Table of Content

Table of Content

Executive Summary

Project Name	Al Mabrook Financials Inc.
Overview	The Al Mabrook contract integrates the real-world assets tokenization standard tokens - ERC3643. Some of the contracts in scope share features of a factory contract that creates clones of the Escrow and Al Mabrook contracts. The creation of contract clones is restricted to the protocol owner, who can afterwards update the admin at the Al Mabrook contract level. Following the requests of investors triggering an action - invest, uninvest, refund - the protocol performs these calls with the admin invoking these smart contract functions. The Al Matoken flows between the admin and the escrow contract.
Timeline	12 January 2024 - 5th March 2024
Update code Received:	4th April
Second Review:	4th April 2024 April to 8th April 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyze the Al Mabrook Contracts for quality, security, and correctness.
Source code	*https://mumbai.polygonscan.com/address/0xfe1fb9A2d1c832E7820aC840c8Ce57f859b7703d -> Alambrook* *->This smart contract has functionality to cancel, withdraw, invest etc.* *https://mumbai.polygonscan.com/address/0xF6a9aA43D57e9b921Df507E8f86a2fffd1BdB980 -> Facotry Al Mabrook* *-> This smart contract is responsible for create a project*

Executive Summary

Source Code

https://mumbai.polygonscan.com/address/0x828305de8df4ac61f36427c874d3678681882ad5 -> token

-> This smart contract is responsible for launch , minting and transfer,block ,unblock and etc

https://mumbai.polygonscan.com/address/0x43a96e5A786a7D6C638E4Bc06a2565d4D70FCB27 -> Refund Moderator

-> This smart contract is responsible for refund

Fixed in

Branch-updated_8thMarch
f332117486ac0c9f936367e0676bfb356dd25458

Number of Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	2	1	0	4

Checked Vulnerabilities

We scanned the application for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- Transaction-Ordering Dependence
- Redundant fallback function
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Hardhat, Foundry.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Issues Found

High Severity Issues

1. Missing critical input validation check and failure to invoke Ownable_init in the init function causes the protocol to lose ownership

Path

Token.sol

Function

init

Description

The Token contract inherits IToken, AgentRoleUpgradeable, and the TokenStorage. When the init function is invoked, all of the state variables would be updated and also the state variables in the inherited contract. In this case, where the Ownable_init() function from the OwnableUpgradeable is not invoked in the init function from the Token contract, the _owner address would be address zero since it was never invoked and it is impossible to invoke afterwards since the OwnableUpgradeable is an abstract contract. This function could have identified the issue if it has employed critical input sanitization to ensure that these addresses are not null addresses.

```
function init(
    address _identityRegistry,
    address _compliance,
    string memory _name,
    string memory _symbol,
    uint8 _decimals,
    address _onchainID
) public initializer {
    //@audit missing input validation      You, 17 minutes ago • Uncommitted changes
    //@audit-issue failure to invoke the Ownable_init function

    tokenName = _name;
    tokenSymbol = _symbol;
    tokenDecimals = _decimals;
    tokenOnchainID = _onchainID;
    tokenIdentityRegistry = IIdentityRegistry(_identityRegistry);
    emit IdentityRegistryAdded(_identityRegistry);
    tokenCompliance = ICompliance(_compliance);
    emit ComplianceAdded(_compliance);
    emit UpdatedTokenInformation(tokenName, tokenSymbol, tokenDecimals, TOKEN_VERSION, tokenOnchainID);
}
```

Proof of Concept:

```
ftrace | funcSig
function testAddAgent() external {
    // make the deployer of the contract to addAgent
    vm.startPrank(owner);

    vm.expectRevert("Ownable: caller is not the owner");
    almabrookToken.addAgent(owner);

    console.log(almabrookToken.owner());
    vm.stopPrank();
}
```

```
[19614] AlmaBrookTest::testAddAgent()
  ├─ [0] VM::startPrank(OWNER: [0x356f394005D3316ad54d8f22b40D02Cd539A4a3C])
  │   └─ ← ()
  ├─ [0] VM::expectRevert(Ownable: caller is not the owner)
  │   └─ ← ()
  ├─ [2679] Token::addAgent(OWNER: [0x356f394005D3316ad54d8f22b40D02Cd539A4a3C])
  │   └─ ← revert: Ownable: caller is not the owner
  ├─ [2487] Token::owner() [staticcall]
  │   └─ ← 0x0000000000000000000000000000000000000000
  ├─ [0] console::log(0x0000000000000000000000000000000000000000) [staticcall]
  │   └─ ← ()
  ├─ [0] VM::stopPrank()
  │   └─ ← ()
  └─ ← ()
```

Recommendation

Invoke the __Ownable_init function and also add sanity checks on input parameters.

Reference

https://github.com/ERC-3643/ERC-3643/blob/main/contracts/token/Token.sol

Status

Resolved

QuillAudits' Comment

Issue is resolved but variables, emits, init functions should be arranged according to the given reference.

Al Mabrook Team Comment

It has been arranged as per the reference.

```
51      */
52      function init(
53          address _identityRegistry,
54          address _compliance,
55          string memory _name,
56          string memory _symbol,
57          uint8 _decimals,
58          address _onchainID
59      ) public initializer {
60          require(owner() == address(0), "already initialized");
61          __Ownable_init();
62          tokenName = _name;
63          tokenSymbol = _symbol;
64          tokenDecimals = _decimals;
65          tokenOnchainID = _onchainID;
66          tokenIdentityRegistry = IIdentityRegistry(_identityRegistry);
67          tokenCompliance = ICompliance(_compliance);
68          emit IdentityRegistryAdded(_identityRegistry);
69          emit ComplianceAdded(_compliance);
70          emit UpdatedTokenInformation(tokenName, tokenSymbol, tokenDecimals, TOKEN_VERSION, tokenOnchainID);
71
```

2. Missing Modifier Restriction on Launch and deployEscrow functions

Path

FactoryAlma.sol , FactoryEscrow.sol

Function

launch , deployEscrow

Description

These are two critical functions to the protocol that allows for the protocol owners to create new instances of the Escrow and Al Mabrook contracts. However, these functions do not have a modifier restriction to allow only privileged addresses to invoke. This implies that malicious users can create multiple Escrow and Al Mabrook contracts for malicious purposes.

Proof of Concept

```
function testAttakcerDeploysMultipleEscrowsAndAlmaBrooks() external {
    address attacker = makeAddr("Attacker_Address");
    vm.startPrank(attacker);
    for (uint256 i = 0; i <= 100; i++) {
        //deploy multiple Escrow contracts
        escrowContract = Escrow(payable(escrowFactory.deployEscrow(address(almabrookToken))));
        escrowInterface = IEscrow(address(escrowContract));

        // deploy multiple  Almabrook contracts
        almaFactory.launch(
            address(almabrookToken), 3000 ether, 30 ether, block.timestamp, block.timestamp, true, escrowInterface
        );
    }
    vm.stopPrank();

}
```

contracts/FactoryAlma.sol:FactoryAlma contract						
Deployment Cost	Deployment Size					
2338813	11621					
Function Name	min	avg	median	max	# calls	
launch	2113701	2113937	2113701	2137601	101	

contracts/FactoryEscrow.sol:FactoryEscrow contract						
Deployment Cost	Deployment Size					
996875	5034					
Function Name	min	avg	median	max	# calls	
deployEscrow	866813	867247	866813	910713	101	

Recommendation

Create a modifier and add to these critical functions. Using the Openzeppelin Ownable2step contract can also be a good solution as it introduces security and flexibility. The security provides the onlyOwner modifier which can restrict the functions to only privileged addresses and flexibility is the aspect of transferring ownership safely between addresses, if the need could arise.

Status

Resolved

QuillAudits' Comment

While deploying the Al Ma contract through AlmaFactory creator and admin both will be set to msg.sender which does not justify the mitigation as only one entity will be entitled to handle the contract. And the functionality in escrow contract where if else is used won't matter as the creator and admin are the same. According to the protocol, the creator can be different from the owner while launching the contract. But if the launch() function is called by the admin both admin and creator will be the same.

Al MaBrook Team Comment

launch function has restriction that only Admin can invoke the function on behalf of Project Owner. Since we allow Project Owners to use Al Mabrook token as token for the project, Creator and Admin should remain the same because Platform Admin is the Creator of Al Matoken.

QuillAudits' Comment on 2nd Review

The creator will be the same as the admin after launch. Then admin updates admin of the contract to creator with the updateAdmin

Medium Severity Issues

1. Some new projects may end one second after launch time

Path

FactoryAlma.sol/ contracts/FactoryAlma.sol

Function

launch

Description

```
    require(_startAt >= block.timestamp, "Invalid start time");
    //@audit-issue some new projects can add a second after launch time
    // if projects are incorrectly launched when _startAt == _endAt
    require(_endAt >= _startAt, "Invalid end time");

    _campaignId.increment();
    uint campaignId = _campaignId.current();

    addr = address(
        new AlmaBrook(
            campaignId,
            _token,
            msg.sender,
            owner,
            _target,
            _softCap,
            _startAt,
            _endAt,
            _almaToken,
            _escrowContract
        )
    );
```

The creation of Al Mabrook contract clone is done with the launch function. This function checks some key properties that make up the Al Mabrook contract. While the first check appropriately ensures that _startAt can either be equal to the block.timestamp or greater, this implies that the Al Mabrook activities should begin immediately after launch or after a period of time set in the future. However, the second check ensures that _endAt time can be equal to the _startAt or greater than. This is where the vulnerability arises. If at any point in time, _endAt == _startAt, the project ends after one second.

Proof of Concept (Add POC if Applicable)

```
ftrace | funcSig
function testLaunchAndProjectEndsInOneSecond() external {
    // deploy an escrow and an almabrook contract
    address oneEscrow = escrowFactory.deployEscrow(address(almabrookToken));

    escrowContract = Escrow(payable(oneEscrow));
    escrowInterface = IEscrow(address(escrowContract));

    // launch an almabrook contract through the factory
    address oneAlma = almaFactory.launch(address(almabrookToken), 3000 ether, 30 ether, block.timestamp, block.timestamp, true,
    escrowInterface);

    almabrookContract = AlmaBrook(oneAlma);

    vm.startPrank(owner);
    almabrookContract.updateAdmin(admin);
    vm.stopPrank();

    vm.warp(block.timestamp + 1 seconds);

    Order memory c = Order(10 ether, 1, 1, address(1), address(0), oneEscrow, FundingType.Fiat);

    vm.startPrank(admin);
    vm.expectRevert("The project has ended");
    almabrookContract.invest(c);
    vm.stopPrank();

}
```

Recommendation

Change the “require” check so that _endAt is greater than _startAt, always after launch.

```
--      require(_endAt⬆ >= _startAt⬆, "Invalid end time");
++      require(_endAt⬆ > _startAt⬆, "Invalid end time");
```

Status

Resolved

Low Severity Issues

1. Use the latest T-REX ERC3643 library

Description

The Al Mabrook contracts are using old TREX-ERC3643 library which had issue H1 from high severity issues which would have caused contracts to remain without owner after deployment. The new library has new improvements. So it is advised to use new library contracts.

Recommendation

Use the latest TREX ERC3643 library to integrate into the contract.

Status

Acknowledged

Informational Issues

2. Emit one indexed event to save over 1000 gas

Path

Escrow.sol

Function

deployEscrow

Event

```
    //@audit-issue maintain one emission of indexed event as it saves over 1000 gas.
    emit Deployed(msg.sender, escrowAdd);
    emit Launched(escrowId, msg.sender);
```

Description

The deployEscrow function emits two events which share similar parameters. With the design architecture of the contract, the owner parameter in the Deployed event and the creator in the Launched event will always result in the same address. These two events will increase the gas cost.

Proof of Concept

Two event emission

```
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.86ms
| contracts/FactoryEscrow.sol:FactoryEscrow contract |                 |        |        |        |         |
|----------------------------------------------------|-----------------|--------|--------|--------|---------|
| Deployment Cost                                    | Deployment Size |        |        |        |         |
| 996875                                             | 5034            |        |        |        |         |
| Function Name                                      | min             | avg    | median | max    | # calls |
| deployEscrow                                       | 910713          | 910713 | 910713 | 910713 | 1       |
```

one event emission

```
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.87ms
| contracts/FactoryEscrow.sol:FactoryEscrow contract |                 |        |        |        |         |
|----------------------------------------------------|-----------------|--------|--------|--------|---------|
| Deployment Cost                                    | Deployment Size |        |        |        |         |
| 986062                                             | 4980            |        |        |        |         |
| Function Name                                      | min             | avg    | median | max    | # calls |
| deployEscrow                                       | 909657          | 909657 | 909657 | 909657 | 1       |
```

Average of two events = 910,713

Average of one event = 909,657

Gas saved = 910,713 - 909,657 = 1056

Recommendation

Design function to emit one indexed event when function is invoked

Status

Resolved

3. Double onlyOwner check will increase the cost of gas

Path

Amlabrook.sol

Function

cancel

Event

```
ftrace | funcSig
function cancel(uint96 _campaignId) external onlyOwner returns (bool) {
    require(_campaignId == campaignId, "Invalid project ID");
    require(creator == msg.sender, "OnlyCreator: caller is not the creator");
    // require(block.timestamp < startAt, "The project has already started");
    isCanceled = true;
    emit Canceled(isCanceled);
    return true;
}
```

Description

The cancel function has an onlyOwner modifier that checks that **creator == msg.sender**. After this check, this check was again checked for the second time. The gas cost for calling this function will charge double for this check and invariably increase gas fee.

Recommendation

Remove the second check happening in the cancel function.

Status

Resolved

4. Rename state variable escrowId to be different from individual id in Escrow struct

Path

Escrow.sol

Variable

```
//@audit-issue rename variable to escrowIds or totalEscrowIds.
uint public escrowId;
```

Description

The escrowId is a state variable that tracks all of the created escrows. However, this state variable gets updated every time an escrow is deployed and gets set as the id for an individual escrow.

Proof of Concept (Add POC if Applicable)

```
function testDisplayEscrowId() external {
    // deploy an escrow and an almabrook contract
    address oneEscrow = escrowFactory.deployEscrow(address(almabrookToken));
    console.log("after first escrow", escrowFactory.escrowId());
    address twoEscrow = escrowFactory.deployEscrow(address(almabrookToken));
    console.log("after second escrow", escrowFactory.escrowId());
    address thirdEscrow = escrowFactory.deployEscrow(address(almabrookToken));
    console.log("after third escrow", escrowFactory.escrowId());
    address fourthEscrow = escrowFactory.deployEscrow(address(almabrookToken));
    console.log("after fourth escrow", escrowFactory.escrowId());
    address fifthEscrow = escrowFactory.deployEscrow(address(almabrookToken));
    console.log("after fifth escrow", escrowFactory.escrowId());
}
}
```

```
[PASS] testDisplayEscrowId() (gas: 4398066)
Logs:
  after first escrow 1
  after second escrow 2
  after third escrow 3
  after fourth escrow 4
  after fifth escrow 5
```

Recommendation

Rename the escrowId variable to escrowIds or totalEscrowIds for better clarification.

Status

Resolved

5. Public function should be marked as external to save gas

Path

Almabrook.sol

Variable

```
ftrace | funcSig
function getBalance(address sender⬆) public view returns (uint256) {
    uint256 bal = token.balanceOf(sender⬆);
    return bal;
}
```

Description

The getBalance function is a function in Al Mabrook declared as public to get the balance of an address. However, this function was not invoked within the contract. In order to save gas invoking this function, it is recommended to mark this function as external.

Recommendation

Mark this function external to save gas.

Status

Resolved

Functional Tests

Some of the tests performed are mentioned below:

Token contract:

- Should setUp the token contract and identify identity registry contract, claim topics registry, and the identity registry storage - setting up token deployment and after deploying components of the ERC3643 token.
- Should revert if the owner of the token after deployment is address zero - this test case checks to know if ownership is maintained before and after deployment. This inconsistency helped discover the H1 issue.
- Should mint the tokens to a verified identities address - this scenario checks how the tokens are minted to verified identity addresses.
- Should affirm the approval and allowances features - test how approval of the tokens to a spender is performed and how the spender spends his allowance.
- Should test who can invoke the transferFrom function of the token - this test and the ones before test the ERC20 compatibility features related to the tokens. This checks how transferFrom function is invoked

Al Mabrook and Escrow Contracts:

- Should integrate the ERC3643 token into the Al Mabrook infrastructure - should deploy the Al Mabrook contract with using the Al Matoken and created escrow contract
- Should test that admins can be updated by the contract creator - Since contract owner alone can trigger the updateAdmin function, this tests that the owner can set a new address as the admin.
- Should confirm that on the invocation of invest function, tokens are sent into escrow - when users make request to invest and admin invoke the invest function, expected amount of Al Matoken for users are sent into the escrow contract.
- Should revert when admin has insufficient tokens in their balance to invoke invest - this test verifies that admin who will often trigger the invest function should always have enough token to get a successful transaction.
- Should uninvest successfully and tokens in escrow contract returned to the admin - after the request of users to uninvest, admin should trigger the uninvest functionality and the Al Matoken is returned from the Escrow to the admin address.

Functional Tests

- Should revert if the uninvest function is triggered after 3 days of investment - should revert if users make an uninvest request after 3 days of investment.
- Should revert if wrong order is set for the user while investing - should revert if wrong order info is passed by admin for a right request.
- Should be able to run withdrawByInvestor function - should be able to withdraw Al Matoken from escrow to admin when there's a request for withdrawByInvestor.
- Should revert on invalid project id.
- Should revert on invalid escrow wallet - should revert when a project is initialized with a wrong escrow contract
- Should be able to call refund function - should refund Al Mabrook tokens when refund
- Should be able to transfer tokens to user after project completion - this test case affirms the possibility of tokens transfer to users after project completion

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the **Al Mabrook** codebase. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. In the end, the Al Mabrook Team resolved all issues and acknowledged one low issue.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in **Al Mabrook** smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of **Al Mabrook** smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the **Al Mabrook** to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
April, 2024

For

Mabrook

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com