Security Audit Report

Poolz DelayVault

11/2/2023

Revision: 28/3/2023

PREPARED FOR:
Poolz, Poolz.Finance

ARCADIA CONTACT INFO
Email: audits@arcadiamgroup.com

Telegram: https://t.me/thearcadiagroup

Table of Contents

Executive Summary

1. Introduction and Audit Scope

A Representative Party of POOLZ ("**CLIENT**") engaged The Arcadia Group ("Arcadia"), a software development, research, and security company, to conduct a review of the following Poolz smart contracts on the **The-Poolz/DelayVault** github repository at Commit **#89c0f308c1afc34636ed6c3327c08f892e19a453**

The scope of this audit included the following files:

Contracts	Lines	nLines	nSLOC	Comment Lines	Complex. Score
DelayVault/contracts/DelayVault.sol	95	84	72	10	35
DelayVault/contracts/DelayEvents.sol	21	21	18	2	1
DelayVault/contracts/DelayView.sol	82	57	49	2	55
DelayVault/contracts/DelayData.sol	26	26	21	3	6
DelayVault/contracts/ERC20Token.sol	5	5	2	2	
DelayVault/contracts/DelayModifiers.sol	85	76	60	3	17
DelayVault/contracts/DelayManageable.sol	91	67	56	3	40

Another review was also done after the changes on **DelayVault v1.1.0** at commit hash **#03490e948a05f6556d0571e8781547ec42387ed1**

Another review was also done after the changes on **DelayVault v1.1.3** at commit hash **#3c0f2e003b96ed54f4aa99747ac22d3c27a6ed08**

Another review was also done after the changes on **DelayVault v1.2.1** at commit hash **#8a75232861aacc3862e4718e959792308511766a**

2. Audit Summary

a. Audit Methodology

Arcadia completed this security review using various methods primarily consisting of dynamic and static analysis. This process included a line-by-line analysis of the in-scope contracts, optimization analysis, analysis of key functionalities and limiters, and reference against intended functionality.

The followings are the steps we have performed while auditing the smart contracts:

- Investigating the project and its technical architecture overview through its documentation
- Understanding the overview of the smart contracts, the functions of the contracts, the inheritance, and how the contracts interface with each others thanks to the graph created by Solidity Visual Developer
- Manual smart contract audit:
    - Review the code to find any issue that could be exploited by known attacks listed by Consensys
    - Identifying which existing projects the smart contracts are built upon and what are the known vulnerabilities and remediations to the existing projects
    - Line-by-line manual review of the code to find any algorithmic and arithmetic related vulnerabilities compared to what should be done based on the project's documentation
    - Find any potential code that could be refactored to save gas
    - Run through the unit-tests and test-coverage if exists
- Automated smart contract audit:
    - Scanning for vulnerabilities in the smart contracts using Static Code Analysis Software
    - Making a static analysis of the smart contracts using Slither
- Additional review: a follow-up review is done when the smart contracts have any new update. The follow-up is done by reviewing all changes compared to the audited commit revision and its impact to the existing source code and found issues.

b. Summary

There were **6** issues found, **0** of which were deemed to be ‘critical’, and **1** of which were rated as ‘high’.

Severity Rating	Number of Original Occurrences	Number of Remaining Occurrences
CRITICAL	0	0
HIGH	1	0
MEDIUM	3	0
LOW	1	1
INFORMATIONAL	1	0

Findings in Manual Audit

1. Lack of reentrancy checks

Issue ID

DV-1

Risk level

Severity: High, likelihood: Medium

Code segment

```
function CreateVault(
       address _token,
       uint256 _amount,
       uint256 _startDelay,
       uint256 _cliffDelay,
       uint256 _finishDelay
   ) external whenNotPaused notZeroAddress(_token)
isTokenActive(_token) {
   //..
   TransferInToken(_token, msg.sender, _amount);
}

function Withdraw(address _token) external isVaultNotEmpty(_token) {
   //..
   TransferToken(_token, msg.sender, lockAmount);
}
```

Description

The functions `CreateVault` and `Withdraw` lack re-entrancy guards that could be exploited in `TransferInToken` and `TransferToken`.

Code location

```
contracts/DelayVault.sol
```

Proof of concept

–

Recommendation

Add `ReentrancyGuard` of OpenZeppelin for the two functions.

2. Function `_isTokenActive` unclarified

Issue ID

DV-2

Risk level

Severity: Informational

Code segment

```
modifier isTokenActive(address _token) {
        _isTokenActive(_token);
        _;
    }

function _isTokenActive(address _token) private view {
    require(
        DelayLimit[_token].isActive,
        "there are no limits set for this token"
    );
}
```

Description

The function `_isTokenActive` is redundant and its purpose can be merged into the modifier.

Code location

```
contracts/DelayModifiers.sol
```

Proof of concept

_

Recommendation

The modifier isTokenActive can be simply refined as following:

```
modifier isTokenActive(address _token) {
        require(
            DelayLimit[_token].isActive,
            "there are no limits set for this token"
        );
        _;
    }
}
```

3. Function `setMinDelays` shouldnt require ordered amounts

Issue ID

DV-3

Risk level

Severity: Medium, Likelihood: Medium

Code segment

```
// file DelayManageable.sol
function setMinDelays(
       address _token,
       uint256[] calldata _amounts,
       uint256[] calldata _startDelays,
       uint256[] calldata _cliffDelays,
       uint256[] calldata _finishDelays
   ) external onlyOwnerOrGov notZeroAddress(_token) {
       {
           // Stack Too deep error fixing
           _equalValues(
               _amounts.length,
               _startDelays.length,
               _cliffDelays.length,
               _finishDelays.length
           );
           _orderedArrays(_amounts, _startDelays, _cliffDelays,
_finishDelays);
       }
}

// file DelayVault.sol
(
            uint256 _startMinDelay,
            uint256 _cliffMinDelay,
            uint256 _finishMinDelay
```

```
) = GetMinDelays(_token, _amount);
        {
            // Checking the minimum delay for each timing parameter.
            _checkMinDelay(_startDelay, _startMinDelay);
            _checkMinDelay(_cliffDelay, _cliffMinDelay);
            _checkMinDelay(_finishDelay, _finishMinDelay);
        }

// file DelayView.sol
function GetMinDelays(address _token, uint256 _amount)
       public
       view
       isTokenActive(_token)
       returns (
           uint256 _startDelay,
           uint256 _cliffDelay,
           uint256 _finishDelay
       )
   {
       Delay memory delayLimit = DelayLimit[_token];
       if (delayLimit.Amounts.length == 0 || delayLimit.Amounts[0] >
_amount)
           return (0, 0, 0);
       _startDelay = delayLimit.StartDelays[0];
       _cliffDelay = delayLimit.CliffDelays[0];
       _finishDelay = delayLimit.FinishDelays[0];
       for (uint256 i = 0; i < delayLimit.Amounts.length; i++) {
           if (_amount >= delayLimit.Amounts[i]) {
               _startDelay = delayLimit.StartDelays[i];
               _cliffDelay = delayLimit.CliffDelays[i];
               _finishDelay = delayLimit.FinishDelays[i];
           } else {
               break;
           }
       }
   }
```

Description

The function `setMinDelays` requires that input parameters `_amounts` is an ordered array along with `_startDelays`, `_cliffDelays`, and `_finishDelays`. However, for a token release, it is not logical to require that a higher delay would be corresponding to a higher unlocked token amount.

This issue should also be addressed in the function `CreateVault`, which calls the function GetMinDelays that is based on the input parameter `amount`.

Code location

```
contracts/DelayManageable.sol
```

Proof of concept

–

Recommendation

There is no need to require the `_amounts` parameter to be ordered as the timing parameters. The `_orderedArrays` function should only check the timing parameters `_startDelays`, `_cliffDelays`, and `_finishDelays` as ordered arrays.

4. Duplicates in user's token list

Issue ID

DV-4

Risk level

Severity: Medium, Likelihood: High

Code segment

```
// file DelayVault.sol
       TransferInToken(_token, msg.sender, _amount);
       vault.StartDelay = _startDelay;
       vault.CliffDelay = _cliffDelay;
       vault.FinishDelay = _finishDelay;
       if (!Array.isInArray(Users[_token], msg.sender)) {
           Users[_token].push(msg.sender);
       }
       MyTokens[msg.sender].push(_token);
       emit VaultValueChanged(
           _token,
           msg.sender,
           vault.Amount += _amount,
           _startDelay,
           _cliffDelay,
           _finishDelay
       );

// file DelayView.sol
function GetAllMyTokens(address _user)
        external
        view
        returns (address[] memory)
    {
        return MyTokens[_user];
    }
```

```
    function GetMyTokens(address _user) external view returns
(address[] memory) {
        address[] storage allTokens = MyTokens[_user];
        address[] memory tokens = new address[](allTokens.length);
        uint256 index;
        for (uint256 i = 0; i < allTokens.length; i++) {
            if (VaultMap[allTokens[i]][_user].Amount > 0) {
                tokens[index++] = allTokens[i];
            }
        }
        return Array.KeepNElementsInArray(tokens, index);
    }
```

Description

The `CreateVault` function does not check whether the array `MyTokens[msg.sender]` contains the input `_token` or not, which can result in duplicates in the token array, which can result in incorrect return values in functions GetAllMyTokens and GetMyTokens

Code location

```
contracts/DelayVault.sol
contracts/DelayView.sol
```

Proof of concept

_

Recommendation

The `CreateVault` function should check whether the input `_token` is in the list of token array of the user.

5. Gas optimizations

Issue ID

DV-5

Risk level

Severity: Low, Likelihood: High

Code segment

```
// file DelayVault.sol
if (!Array.isInArray(Users[_token], msg.sender)) {
    Users[_token].push(msg.sender);
}

// file DelayView.sol
for (uint256 i = 0; i < delayLimit.Amounts.length; i++) {
            if (_amount >= delayLimit.Amounts[i]) {
                _startDelay = delayLimit.StartDelays[i];
                _cliffDelay = delayLimit.CliffDelays[i];
                _finishDelay = delayLimit.FinishDelays[i];
            } else {
                break;
            }
        }
```

Description

- In `DelayVault.sol` file, it can be optimized by using a mapping for the `_token` and `msg.sender` for easier check of existence
- `delayLimit.Amounts.length` could be replaced by using a parameter for the `length` to avoid re-reading length in every iteration.

Code location

```
contracts/DelayVault.sol
contracts/DelayView.sol
```

Proof of concept

–

Recommendation

As above in Description section

Automated Audit

Static Analysis with Slither

We run a static analysis against the source code using Slither, which is a Solidity static analysis framework written in Python 3. Slither runs a suite of vulnerability detectors, prints visual information about contract details. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses.

The following shows the results found by the static analysis by Slither. We reviewed the results, and, except the Reentrancy issues that are identified as above, all of the other issues found by Slither are false positives.

```
ERC20Helper.TransferToken(address,address,uint256) (poolz-helper-v2/contracts/ERC20Helper.sol#22-34)
ignores return value by ERC20(_Token).transfer(_Reciver,_Amount)
(poolz-helper-v2/contracts/ERC20Helper.sol#29)
ERC20Helper.TransferInToken(address,address,uint256) (poolz-helper-v2/contracts/ERC20Helper.sol#44-57)
ignores return value by ERC20(_Token).transferFrom(_Subject,address(this),_Amount)
(poolz-helper-v2/contracts/ERC20Helper.sol#51)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

ETHHelper.TransferETH(address,uint256) (poolz-helper-v2/contracts/ETHHelper.sol#35-43) uses a dangerous
strict equality:
        - require(bool,string)((beforeBalance + _amount) == address(_Reciver).balance,The transfer did not
complite) (poolz-helper-v2/contracts/ETHHelper.sol#39-42)
ERC20Helper.TransferInToken(address,address,uint256) (poolz-helper-v2/contracts/ERC20Helper.sol#44-57) uses
a dangerous strict equality:
        - require(bool,string)((OldBalance + _Amount) == CheckBalance(_Token,address(this)),recive wrong
amount of tokens) (poolz-helper-v2/contracts/ERC20Helper.sol#53-56)
ERC20Helper.TransferToken(address,address,uint256) (poolz-helper-v2/contracts/ERC20Helper.sol#22-34) uses a
dangerous strict equality:
        - require(bool,string)((CheckBalance(_Token,address(this)) + _Amount) == OldBalance,recive wrong
amount of tokens) (poolz-helper-v2/contracts/ERC20Helper.sol#30-33)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities


DelayView.GetMyTokens(address).index (DelayView.sol#31) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC20Helper.ApproveAllowanceERC20(address,address,uint256)
(poolz-helper-v2/contracts/ERC20Helper.sol#59-66) ignores return value by
ERC20(_Token).approve(_Subject,_Amount) (poolz-helper-v2/contracts/ERC20Helper.sol#65)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

GovManager.setGovernerContract(address) (poolz-helper-v2/contracts/GovManager.sol#18-20) should emit an
event for:
        - GovernerContract = _address (poolz-helper-v2/contracts/GovManager.sol#19)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

GovManager.setGovernerContract(address)._address (poolz-helper-v2/contracts/GovManager.sol#18) lacks a
zero-check on :
                - GovernerContract = _address (poolz-helper-v2/contracts/GovManager.sol#19)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in DelayVault.CreateVault(address,uint256,uint256,uint256,uint256) (DelayVault.sol#12-55):
        External calls:
        - TransferInToken(_token,msg.sender,_amount) (DelayVault.sol#39)
                - ERC20(_Token).transferFrom(_Subject,address(this),_Amount)
(poolz-helper-v2/contracts/ERC20Helper.sol#51)
        State variables written after the call(s):
        - MyTokens[msg.sender].push(_token) (DelayVault.sol#46)
        - Users[_token].push(msg.sender) (DelayVault.sol#44)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

Unit Test Coverage

```
Contract: DelayVault
  ✔ should revert invalid start delay (270ms)
  ✔ should revert invalid finish delay (150ms)
  ✔ should revert invalid cliff delay (145ms)
  ✔ should create vault (181ms)
  ✔ should revert shorter blocking period than the last one (245ms)
  ✔ should revert when empty vault (104ms)
  ✔ should revert zero amount (147ms)
  ✔ withdraw tokens when no locked deal (243ms)

Contract: Delay vault admin settings
  ✔ should pause contract (201ms)
  ✔ should set LockedDeal
  ✔ should set min delays (44ms)
  ✔ should revert arrays with dirrent lengths (121ms)
  ✔ should revert with the same value
  ✔ should revert when no limits are set for this token (84ms)
  ✔ should deactivate/activate token (194ms)

Contract: Delay vault data
  ✔ should get delay limit (48ms)
  ✔ get limit delays (85ms)
  ✔ should revert when not ordered amount
  ✔ should revert when not ordered start delays
  ✔ should revert when not ordered finish delays (38ms)
  ✔ should get my token addresses (347ms)
  ✔ should return all data (1219ms)
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	97.53	87.5	100	97.73	
DelayData.sol	100	100	100	100	
DelayEvents.sol	100	100	100	100	
DelayManageable.sol	100	100	100	100	
DelayModifiers.sol	100	90	100	100	
DelayVault.sol	93.33	83.33	100	93.55	69,70
DelayView.sol	100	83.33	100	100	
ERC20Token.sol	100	100	100	100	
All files	97.53	87.5	100	97.73	

Except the function `Withdraw` of `DelayVault` contract, all other contracts are 100% covered by the unit tests.

The uncovered code is as follow:

```
// file DelayVault.sol
if (LockedDealAddress != address(0)) {
           ApproveAllowanceERC20(_token, LockedDealAddress,
lockAmount);
           ILockedDealV2(LockedDealAddress).CreateNewPool(
               _token,
               startDelay,
               cliffDelay,
               finishDelay,
               lockAmount,
               msg.sender
           );
       }
```

As this is an important test to verify whether users can create a locked pool by calling the `Withdraw` function. It is therefore highly recommended to create a test-case to cover this code section.

Issue ID

DV-6

Risk level

Severity: Medium, Likelihood: Medium

Description

- Missing unit-tests to cover the case when `LockedDealAddress` is not null and users are able to create locked pools by calling the `Withdraw` function.

Code location

```
contracts/DelayVault.sol
```

Recommendation

Write additional test-cases to cover the uncovered code.

Review of DelayVault v1.1.0 at commit hash

#03490e948a05f6556d0571e8781547ec42387ed1

The issue remains unresolved. It is recommended that the team should write a unit-test case for this case.

Recommendations

We notice that the new Poolz token **POOLX** contract has a mint function that could generate new tokens. It is highly recommended to set the only minter as a multisignature contract or a timelock contract to avoid any risks associated with leaking the private key if a minter is an EOA.

The team already set the current only minter of POOLX as the multisignature contract at https://etherscan.io/address/0x3e4588c3C4E6ff3da84ab5401490d9c9eA820d3E#code

Conclusion

Arcadia identified issues that occurred at hash **#89c0f308c1afc34636ed6c3327c08f892e19a453**.

An additional review of the DelayVault contracts were also conducted at v1.1.0 at commit hash **#03490e948a05f6556d0571e8781547ec42387ed1** and at v1.1.3 at commit hash **#3c0f2e003b96ed54f4aa99747ac22d3c27a6ed08.** No issues were introduced with the changed code. Furthermore, 5 of 6 issues were also resolved. There remains only one issue with the unit-test to cover one branch in the code, which was marked as low severity.

An additional review of the DelayVault contracts were also conducted at v1.2.1 at commit hash **#8a75232861aacc3862e4718e959792308511766a.** No issues were introduced with the changed code

Disclaimer

While best efforts and precautions have been taken in the preparation of this document, The Arcadia Group and the Authors assume no responsibility for errors, omissions, or damages resulting from the use of the provided information. Additionally, Arcadia would like to emphasize that the use of Arcadia's services does not guarantee the security of a smart contract or set of smart contracts and does not guarantee against attacks. One audit on its own is not enough for a project to be considered secure; that categorization can only be earned through extensive peer review and battle testing over an extended period.