SOLIDITYLAB

SMART CONTRACT SECURITY AUDIT OF

Summary

Audit Firm Solidity Lab

Prepared By 0x4non, Kiki_Dev, Willboy

Client Firm Raisin Labs

Final Report Date Feb 19th 2023

Audit Summary

RaisinLabs engaged Solidity Lab to review the security of its Smart Contract system. From February 5th to February 17th a team of 3 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Notice that the examined smart contracts are not resistant to external/internal exploit. For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

✅ Verify the authenticity of this report in Solidity Lab's Portfolio.

Table of Contents

Project Overview

Project Name	RaisinLabs
Language	Solidity
Codebase	https://github.com/crypdoughdoteth/RaisinLabs/blob/main/src/Raisin.sol#L184
Commit	5b29f55

Delivery Date	February 19th 2023
Audit Methodology	Static Analysis, Manual Review

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
High	0	0	0	0	0	0
Medium	6	6	0	0	0	0
Low	5	5	0	0	0	0

Audit Scope & Methodology

Scope

ID	File	SHA-1 Checksum
RS	Raisin.sol	26d8c7c46e12024a3cc3358100a0f589ba0f4bef

Methodology

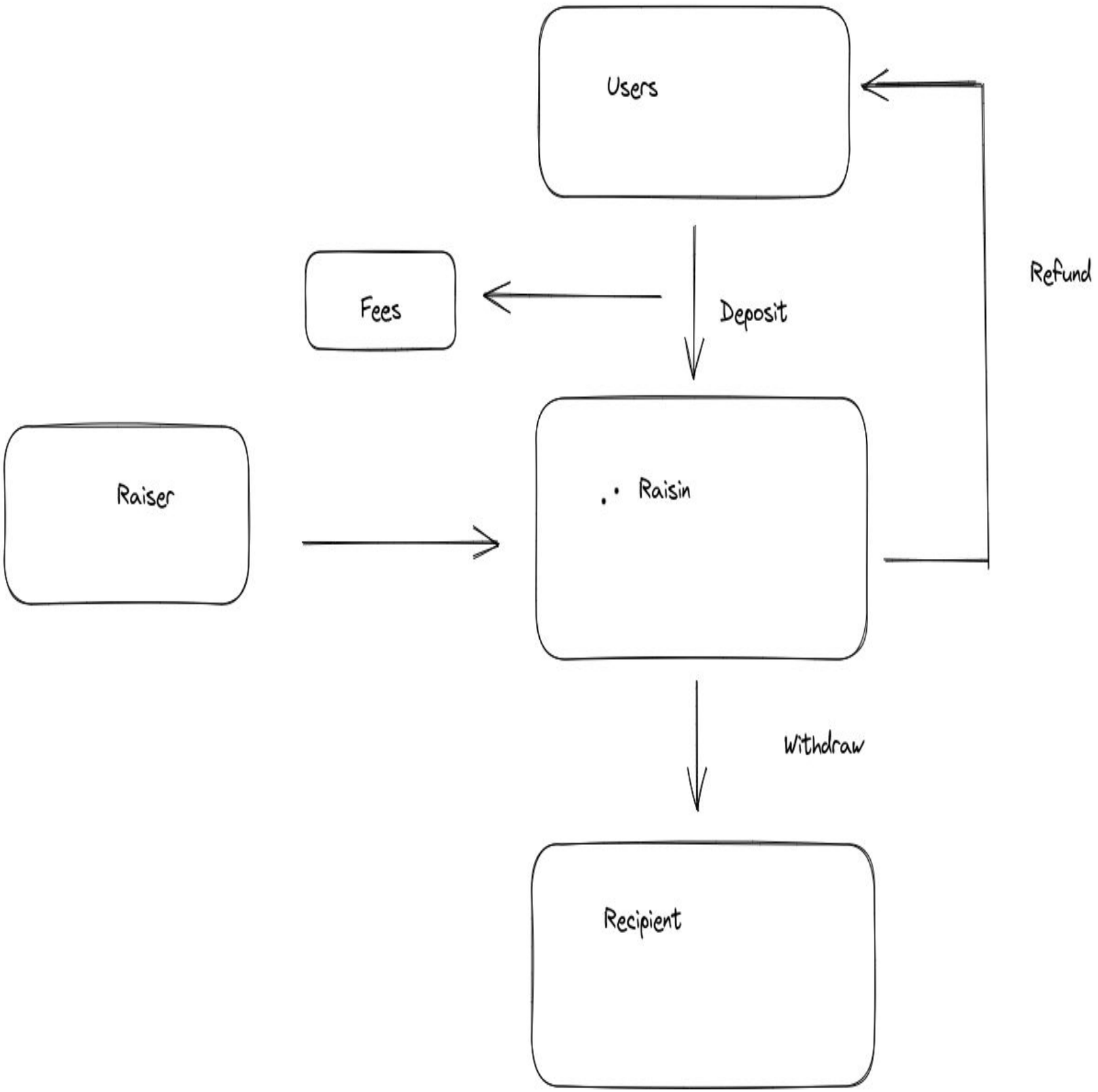
The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by aspiring auditors.

Vulnerability Classifications

Vulnerability Level	Classification
🔴 Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
🟠 High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
🟡 Medium	Inherent risk of future exploits that may or may not impact the smart contract execution.
🟢 Low	Minor deviation from best practices.

Protocol Graph



Findings & Resolutions

ID	Title	Category	Severity	Status
M-1	Tokens that have a “fee on transfer” will break the protocol	Token integration	Medium	Pending
M-2	Tokens that generate interest will get stuck	Token integration	Medium	Pending
M-3	A partner cannot have 0 percent feel	Logic error	Medium	Pending
M-4	Ether can get stuck	Trapped Ether	Medium	Pending
M-5	Use safeTransfer and safeTransferFrom instead of transfer and transferFrom	Token integration	Medium	Pending
M-6	Wrong token handling	Token integration	Medium	Pending
L-1	No event emission to critical functions	Events	Low	Pending
L-2	No check that parameter token is a contract	Validation	Low	Pending
L-3	Missing fee bounds could DoS the protocol	DoS	Low	Pending
L-4	Event will be not emit	Events	Low	Pending
L-5	Everyone is able to call fundWithdraw()	Access Control	Low	Pending

M-1 | Tokens that have a “fee on transfer” will break the protocol

Category	Severity	Location	Status
Token integration	Medium	Raisin.sol	Pending

Description

The protocol will break when trying to deal with tokens that has a fee on transfer.

Recommendation

Before transfer tokens from user track the amount of token in contract, and after transfer check how many tokens the contract has received, thats the amount that should be indeed to the user.

M-2 | Tokens that generate interest will get stuck

Category	Severity	Location	Status
Token integration	Medium	Raisin.sol	Pending

Description

If a yield bearing token is whitelisted, the interest that the token will create when it's inside the contract will be not sending to the recipient. It will be stuck inside the contract.

Recommendation

Consider whether interest bearing tokens should be supported and add logic to transfer interest to the user if they are.

M-3 | A partner cannot have 0 percent fee

Category	Severity	Location	Status
Centralization / Privilege	Medium	Raisin.sol: 234-L235	Pending

Description

If the partner has a 0 percent fee `pf` will be 0, and it will calculate amount with the default fee value

Recommendation

Use a struct to identify the partnership, use a bool to know if is a active partner and a uint16;

```
struct Partner {
  uint16 fee;
  bool active;
}
mapping (address => Partner) private partnership

function calculateFee(uint amount, adderess raiser) private view returns (uint _fee){
  if(partnership[raiser].active) {
    _fee = (amount * partnership[raiser].fee) / 10000
  } else {
    _fee = (amount * fee) / 10000;
  }
}
```

M-4 | Ether can get stuck

Category	Severity	Location	Status
Centralization / Privilege	Medium	Raisin.sol: 150, 170 181	Pending

Description

Some functions are payable but don’t use `msg.value` and the contract has no method to withdraw stuck ether, this means that if someone sends ether by mistake this will get stuck forever.

Recommendation

Remove `payable`.

M-5 | Use `safeTransfer` and `safeTransferFrom` instead of `transfer` and `transferFrom`

Category	Severity	Location	Status
Token integration	Medium	Raisin.sol: 215	Pending

Description

Some non standard tokens may not return any value.

Recommendation

Use safeTransfer lib, you could use OpenZeppelin or Solmate implementations.

M-6 | Wrong token handling

Category	Severity	Location	Status
Token transfer	Medium	Raisin.sol: 176-177 188-189	Pending

Description

Currently you are doing and approve to itself, this not necessary and should be avoided.

Recommendation

```
diff --git a/src/Raisin.sol b/src/Raisin.sol
index 72eca09..1b31675 100644
--- a/src/Raisin.sol
+++ b/src/Raisin.sol
@@ -153,8 +153,8 @@ contract RaisinCore is Ownable {
         uint donation = amount - calculateFee(amount, msg.sender);
         donorBal[msg.sender][index] += donation;
         raisins[index]._fundBal += donation;
-        erc20Transfer(token, msg.sender, vault, (amount - donation));
-        erc20Transfer(token, msg.sender, address(this), donation);
+        token.safeTransfeFrom(msg.sender, vault, (amount - donation));
+        token.safeTransfeFrom(msg.sender, address(this), donation);
         emit TokenDonated (msg.sender, token, donation, index);

     }
@@ -173,8 +173,7 @@ contract RaisinCore is Ownable {
         uint bal = raisins[index]._fundBal;
         raisins[index]._fundBal = 0;
         IERC20 token = raisins[index]._token;
-        approveTokenForContract(token, bal);
-        erc20Transfer(token, address(this), raisins[index]._recipient, bal);
+        token.safeTransfer(raisins[index]._recipient, bal);
         emit FundEnded(index);
     }

@@ -185,37 +184,10 @@ contract RaisinCore is Ownable {
         donorBal[msg.sender][index] -= bal;
         raisins[index]._fundBal -= bal;
         IERC20 token = raisins[index]._token;
-        approveTokenForContract(token, bal);
-        erc20Transfer(token, address(this), msg.sender, bal);
+        token.safeTransfer(msg.sender, bal);
         if (bal == 0){emit FundEnded(index);}
     }

-    /* ////////////////////////////////////////////////////////////////////////
-    /                                                                        /
-    /                                                                        \
-    /                        External Interactions                           /
-    /                                                                        \
-    /                                                                        /
-    ////////////////////////////////////////////////////////////////////////*/
-
-    function approveTokenForContract (
-        IERC20 token,
-        uint amount
-    ) private {
-        bool sent = token.approve(address(this), amount);
-        if(!sent){revert notSent();}
-    }
-
-    function erc20Transfer (
-        IERC20 token,
-        address sender,
-        address recipient,
-        uint amount
-        ) private {
-        bool sent = token.transferFrom(sender, recipient, amount);
-        if(!sent){revert notSent();}
-    }
-
     /* ////////////////////////////////////////////////////////////////////////
```

LOW-01 | No event emission to critical functions

Category	Severity	Location	Status
Events	Low	Raisin.sol: 153	Pending

Description

There is no event emission when updating/setting a variable inside a function with the modifier onlyOwner.

Recommendation

Add event emission to critical functions / onlyOwner

LOW-02 | No check that parameter token is a contract

Category	Severity	Location	Status
Validation	Low	Raisin.sol: 246	Pending

Description

Firstly, there is a manual check to verify that the address is a smart contract. However, there is a possibility of error when submitting an address.

Recommendation

Add a check in order to verify that parameter token is a smart contract.

LOW-03 | Missing fee bounds could DOS the protocol

Category	Severity	Location	Status
DoS	Low	Raisin.sol: 242	Pending

Description

The owner can change the *fee* to a value that is so high that it can make the line 153 fail.

The admin could set the *fee* to 1000 and then *donation* will be always equal to 0.

Recommendation

Add a range/limit in order to not break the protocol by an error.

LOW-04 | Event will be not emit

Category	Severity	Location	Status
Events	Low	Raisin.sol: 190	Pending

Description

If we want an emit event, we need to have a user with 0 donation for the index (line 184). This is quite illogical because a user call refund if he did a donation.

```
uint bal = donorBal[msg.sender][index];
```

Recommendation

Take the same principle as line 143.

```
if (raisins[index]._fundBal== 0){emit FundEnded(index);}
```

LOW-05 | Everyone is able to call fundWithdraw()

Category	Severity	Location	Status
Access Control	Low	Raisin.sol: 170	Pending

Description

It can be more logical that the only one who can call this function is the raiser.

However, even if it's a random who is calling the function. The funds will go to the recipient.

Recommendation

Add a check that the caller is the raiser.

QA |

- **Users will not be fully refunded**

  If the raisin failed, the user will not be full refunded (the amount that he gives) but the donation (=amount-fees). Is this your assumption ? Maybe take the fees after the raisin period if it is successful ? In order that the user is able to be fully refunded.

- **Use *uint256* instead of *uint***

  Uint is alias for *uint256* but it is preferable to use *uint256*

- **Use constant for static numbers like 10000**

  https://github.com/crypdoughdoteth/RaisinLabs/blob/main/src/Raisin.sol#L235

- **Format BPS, use `100_00` to represent 100%**
- **Use natspec**

  https://docs.soliditylang.org/en/v0.8.17/natspec-format.html

- ***require*() / *revert*() statements should have descriptive reason strings**
- **Unique case**

Some tokens provide airdrop (nft, erc20). Currently raisin has no way of handling these and any airdrop would be stuck. This may be a unique case that should (maybe) not be taken into account.

QA |

- **Functions not used internally could be marked external**

```solidity
File: Raisin.sol

96:     function getAmount(uint index) public view returns (uint){

99:     function getFundBal(uint index) public view returns (uint){

102:     function getToken(uint index) public view returns (IERC20){

105:     function getRaiser(uint index) public view returns (address){

108:     function getRecipient(uint index) public view returns (address){

111:     function getExpires(uint index) public view returns (uint64){

115:     function getLength() public view returns (uint){
```

QA |

- **Centralization Risk for trusted owners**

  Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.

```solidity
File: Raisin.sol

10: contract RaisinCore is Ownable {

225:     function manageDiscount (address partnerWallet, uint newFee)
external onlyOwner {

236:     function changeGlobalExpiry(uint newExpiry) external onlyOwner
returns (uint64){

240:     function changeFee(uint newFee) external onlyOwner {

244:     function whitelistToken (IERC20 token) external onlyOwner {

249:     function removeWhitelist(IERC20 token) external onlyOwner {

253:     function changeVault(address newAddress) external onlyOwner {

```

Gas Optimizations |

- **Set User balance to 0 when refunding. Deducting balance with -= consumes more gas**

  If the raisin failed, the user will not be full refunded (the amount that he gives) but the donation (=amount-fees). Is this your assumption ? Maybe take the fees after the raisin period if it is successful ? In order that the user is able to be fully refunded.

- **Add indexed to variables that you want to query**

  Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

```solidity
File: Raisin.sol

29:     event TokenDonated (address indexed adr, IERC20 token, uint indexed amount, uint index);
```

- **Some conditionals compare variables to a boolean constant**

Boolean constants can be used directly and do not need to be compared to true or false. You can remove the equality to the boolean constant.

Gas Optimizations |

- **Cache variables to avoid `SLOAD`**

  When you are reading more than once from the same variable. Without updating it, it is recommended to use a memory variable. Instead of using the opcode SLOAD at each call. Same pattern could be applied on *donateToken*, *fundWithdraw* and *refund*.

```solidity
    function endFund (uint index) external {
        if (msg.sender != raisins[index]._raiser){revert notYourRaisin(index);}
        raisins[index]._expires = uint64(block.timestamp);
        if(raisins[index]._fundBal == 0){emit FundEnded(index);}
    }
```

Do

```solidity
    function endFund (uint index) external {
        Raisin memory _raisin = raisins[index]
        if (msg.sender != _raisin._raiser){revert notYourRaisin(index);}
        raisins[index]._expires = uint64(block.timestamp);
        if(_raisin._fundBal == 0){emit FundEnded(index);}
    }
```

- Functions Can be marked payable if only the owner has access

  If only the owner has access to a function it can be marked as payable to save gas. There is no risk of user funds being lost because they won't have access to those functions. All functions with the `OnlyOwner` modifier will benefit from this.

Disclaimer

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

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

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

The assessment services provided by Solidity Lab is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Solidity Lab does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Solidity Lab

Solidity Lab is a community of aspiring auditors guided by Guardian Audits.

To learn more, visit https://lab.guardianaudits.com

To view the Solidity Lab audit portfolio, visit https://github.com/GuardianAudits/LabAudits

To book an audit, message https://t.me/guardianaudits