顯示標題和標籤

Audit Firm: Solidity Lab

Client Firm: Poodl Tech

Prepared By: WangChao, said017

Delivery Date: March 7th, 2023

Client Firm engaged Solidity Lab to review the security of its Smart Contract system. From **Begining date** to **March 7th, 2023**, a team of **2** 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.

Project Overview

Project Name	**RaisinLabs**
Language	Solidity
Codebase	https://github.com/poodlTech/tokenAudit (https://github.com/poodlTech/tokenAudit)
Commit	eebe267b3fdd75a82e09cc270b3c046b2c9f2c84 (https://github.com/poodlTech/tokenAudit/tree/eebe267b3fdd75a82e09cc270b3c046b2c9f2c84

Delivery Date	**March 7th, 2023**
Audit Methodology	Static Analysis, Manual Review

Vulnerability Level	**Total**	**Pending**	**Declined**	**Acknowledged**	**Partially Resolved**	**Resol**
Critical	3	3	0	0	0	0
High	1	1	0	0	0	0
Medium	6	6	0	0	0	0
Low	3	3	0	0	0	0

Audit Scope & Methodology

Scope

ID File	SHA-1	Checksum
1	DividendPayingToken.sol	116833f2af8645ac847e35f98cfe13b9a5a6f36c
2	Token.sol	98b399bd3374ab864c65eb6b9725d2529b9b77ad
3	Presale.sol	426dd0300c95954bf281a7b7df9f13bafbd6005d
4	Airdropper.sol	cee4494a4f1ceb3a45ffcfee6c4dbffc6aaa51f4

Methodology

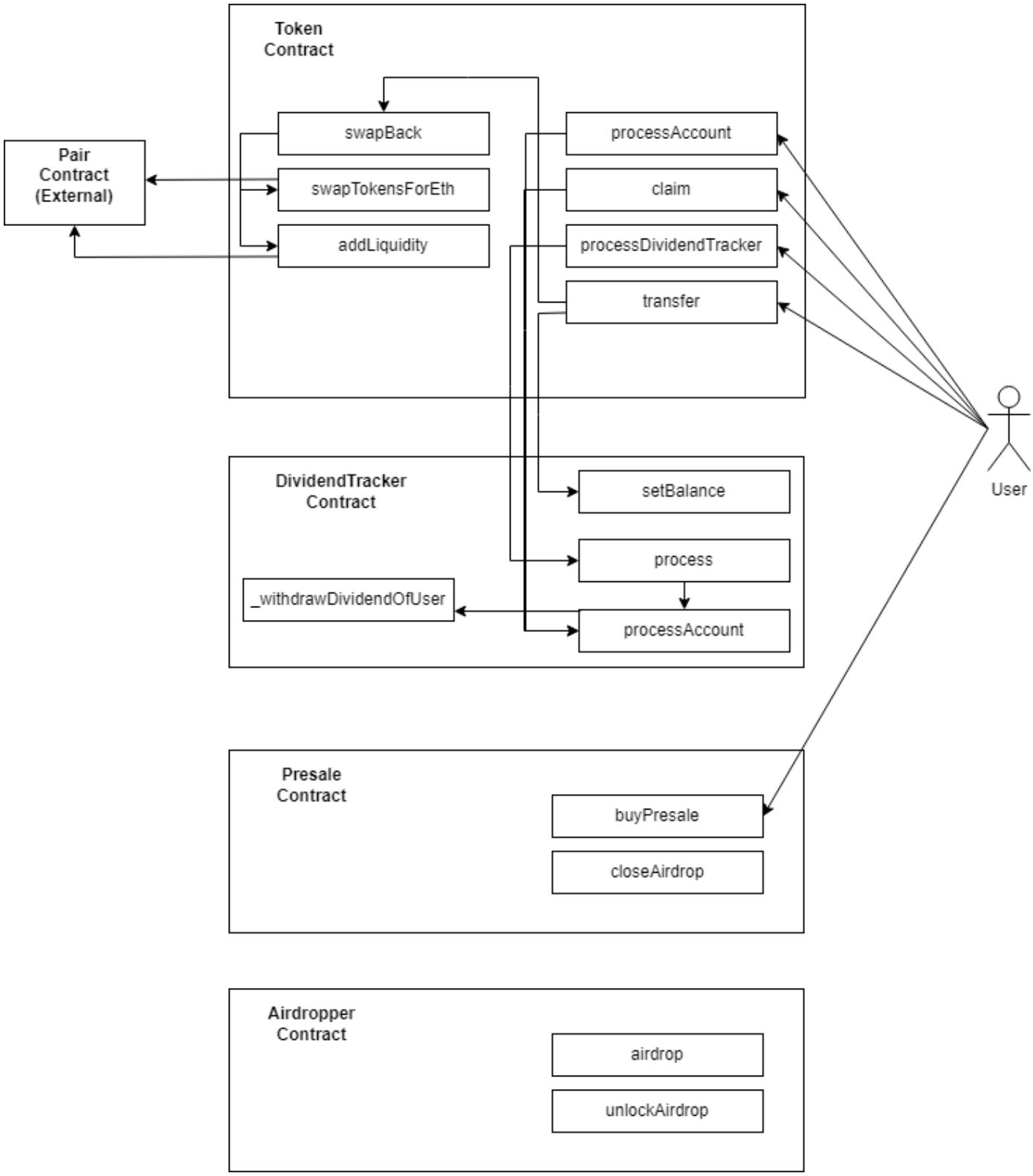
The auditing process pays special attention to the following considerations:

- 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 community 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
C-01	C-01 Incorrect value transferred in `buyPresale` calls.	Logic Error	CRITICAL	Pending
C-02	`boughtAmount` not updated after success `buyPresale` calls.	Logic Error	CRITICAL	Pending
C-03	`claim` and `processAccount` can be called to successfully withdraw user's dividend everytime, ignoring `claimWait` time.	Requirement Violation	CRITICAL	Pending
H-01	Call `buyPresale` directly is not checking Airdrop status.	Requirement Violation	HIGH	Pending
M-01	`updateDividendTracker` and `updateUniswapV2Router` not excluding Token pair and router address from dividends.	Logic error	MEDIUM	Pending
M-02	Unitialized fees inside Token contract constructor.	Logic error	MEDIUM	Pending
M-03	Unitialized `_marketingWalletAddress` inside Token contract constructor.	Logic error	MEDIUM	Pending
M-04	`newToken` address not initialized and cannot be set in Airdropper contract.	Logic error	MEDIUM	Pending
M-05	`token` , `cap` , `exchangeRatio` not initialized and cannot be set in Presale contract.	Logic error	MEDIUM	Pending
M-06	Presale contract doesn't have function to withdraw ether, will locking ether in contract.	Locked Ether	MEDIUM	Pending
L-01	Total fees can higher than `capFees` with maximum up to two times `capFees` value.	Logic Error	LOW	Pending
L-02	`airdrop` function calls run out of gas caused by onbounded arrays.	DOS/Unbounded Array	LOW	Pending
L-03	Missing `payable` modifier for `buyPresale` .	Missed Modifier	LOW	Pending

Critical

C-01 Incorrect value transferred in `buyPresale` calls.

https://github.com/poodlTech/tokenAudit/blob/main/contracts/Presale.sol#L56
(https://github.com/poodlTech/tokenAudit/blob/main/contracts/Presale.sol#L56)

Description:

`buyPresale` transfer `value` instead of `amount` resulting buyer received incorrect amount of token.

Recommendation:

Correct the `transfer` calls using `amount` .

```
IERC20(token).transfer(msg.sender, amount);
```

Resolution:

C-02 `boughtAmount` not updated after success `buyPresale` calls.

https://github.com/poodlTech/tokenAudit/blob/main/contracts/Presale.sol#L51-L58
(https://github.com/poodlTech/tokenAudit/blob/main/contracts/Presale.sol#L51-L58)

Description:

`boughtAmount` is not updated after buyer successfully buy presale token. Buyer can keep repeating `buyPresale` call with `value` <= cap until the token is 0.

Recommendation:

Update `boughtAmount` before transfer token.

Resolution:

C-03 `claim` and `processAccount` can be called to successfully withdraw user's dividend everytime, ignoring `claimWait` time.

https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L256-L258
(https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L256-L258)
https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L260-L262
(https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L260-L262)

Description:

DividendTracker contract define `claimWait` to limit user claiming dividend, once per certain period of time.

But in *Token* contract, `claim` and `processAccount` are defined and calling *DividendTracker*'s `processAccount` without checking the `claimWait`.

Recommendation:

Check the eligibilty by calling `canAutoClaim` first.

```
function claim() external nonReentrant {
    // add checking if can claim this time
    require(dividendTracker.canAutoClaim(lastClaimTimes[msg.sender]), "you
    dividendTracker.processAccount(payable(msg.sender), false);
}

function processAccount(address account) external nonReentrant {
    // add checking if can claim this time
    require(dividendTracker.canAutoClaim(lastClaimTimes[account]), "you hit
    dividendTracker.processAccount(payable(account), false);
}
```

Resolution:

High

H-01 Call `buyPresale` directly is not checking Airdrop status

https://github.com/poodlTech/tokenAudit/blob/main/contracts/Presale.sol#L51-L58
(https://github.com/poodlTech/tokenAudit/blob/main/contracts/Presale.sol#L51-L58)

Description:

In `Presale.sol`, call `buyPresale` directly can bypass presale status check `airDropActive` that checked inside `receive`.

Recommendation:

Move `airDropActive` check from `receive` to `buyPresale`.

Resolution:

Medium

M-01 `updateDividendTracker` and `updateUniswapV2Router` not excluding Token pair and router address from dividends.

https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L113-L125
(https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L113-L125)
https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L127-L140
(https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L127-L140)

Description:

When the owner updating `dividendTracker` by calling `updateDividendTracker` or updating `uniswapV2Router` by calling `updateUniswapV2Router` , it is not excluding Token pair from dividends, also not excluding router address if calling `updateUniswapV2Router` . Will reduce significant amount of dividend rewards earned by users if significant amount of token is provided to liquidity pair contract.

Recommendation:

Call `excludeFromDividends` to remove pair contract and router from dividend calculation.

Resolution:

M-02 Unitialized fees inside Token contract constructor

https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L78-L106
(https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L78-L106)

Description:

None of the fees are initialized inside contructor, this method assume none of transaction happened before the owner set those values. Potentially loss some of dividend from trades until the value is set.

Recommendation:

Initialize the fees inside constructor.

Resolution:

M-03 Unitialized `_marketingWalletAddress` inside Token contract constructor

https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L78-L106
(https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L78-L106)

Description:

`_marketingWalletAddress` is not initialized inside contructor, implication from uninitialized `_marketingWalletAddress` is the `address(_marketingWalletAddress).call{value: amountBNBMarketing}("");` call inside `swapBack()` will burn the value to address 0 until the correct address is set.

Recommendation:

Initialize the `_marketingWalletAddress` inside constructor.

Resolution:

M-04 `newToken` address not initialized and cannot be set in Airdropper contract.

https://github.com/poodlTech/tokenAudit/blob/main/contracts/Airdropper.sol#L33
(https://github.com/poodlTech/tokenAudit/blob/main/contracts/Airdropper.sol#L33)

Description:

`newToken` address not initialized and cannot be set in Airdropper contract. Make all functions involving `newToken` can't work.

Recommendation:

Initialize `newToken` inside constructor.

Resolution:

M-05 `token`, `cap`, `exchangeRatio` not initialized and cannot be set in Presale contract.

https://github.com/poodlTech/tokenAudit/blob/main/contracts/Presale.sol#L33-L35
(https://github.com/poodlTech/tokenAudit/blob/main/contracts/Presale.sol#L33-L35)

Description:

`token`, `cap`, `exchangeRatio` not initialized and cannot be set in Presale contract.

Recommendation:

Initialize `token`, `cap`, `exchangeRatio` inside constructor.

Resolution:

M-06 Presale contract doesn’t have function to withdraw ether, will locking ether in contract.

https://github.com/poodlTech/tokenAudit/blob/main/contracts/Presale.sol (https://github.com/poodlTech/tokenAudit/blob/main/contracts/Presale.sol)

Description:

Presale contract can received ether but doesn’t have function to withdraw ether, will locking ether in this contract.

Recommendation:

add function to withdraw ether.

Resolution:

Low

L-01 Total fees can higher than `capFees` with maximum up to two times `capFees` value.

https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L194-L198 (https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L194-L198)
https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L200-L204 (https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L200-L204)
https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L211-L215 (https://github.com/poodlTech/tokenAudit/blob/main/contracts/Token.sol#L211-L215)

Description:

If the owner setup `sellTopUp` using `setSellTopUp` function, it check `require(value.add(liquidityFee).add(rewardsFee).add(marketingFee) <=capFees),"")` . so the `totalFee + sellTopUp` is <= `capFees` . but if the owner setup the `sellTopUp` fee first while the other fee is 0, he can make it up to 15, and then he calls other fee setup functions `setMarketingFee/setWalletFee/setLiquidityFee` , it doesn’t include checking `sellToUp` in the `require` . then `totalFee + sellToUp` can be > `capFees` with max value 30 (two times `capFees`).

Recommendation:

Include `sellTopUp` check inside `setMarketingFee/setWalletFee/setLiquidityFee` function.

Resolution:

L-02 `airdrop` function calls run out of gas caused by onbounded arrays.

https://github.com/poodlTech/tokenAudit/blob/main/contracts/Airdropper.sol#L47-L56
(https://github.com/poodlTech/tokenAudit/blob/main/contracts/Airdropper.sol#L47-L56)

Description:

`airdrop` function calls can fail accidentally or deliberately and consume large amount of gas.

Recommendation:

Use preferable Merkle tree airdrop solution.

Resolution:

L-03 Missing `payable` modifier for `buyPresale`.

https://github.com/poodlTech/tokenAudit/blob/main/contracts/Presale.sol#L51
(https://github.com/poodlTech/tokenAudit/blob/main/contracts/Presale.sol#L51)

Description:

`buyPresale` use `msg.value` but not add `payable` modifier.

Recommendation:

Add `payable` modifier in `puyPresale`.

Resolution:

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 is a community of practicing auditors guided by Guardian Audits.

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

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