SOLIDIFIED

Audit Report for Taylor Crowdsale. January 22, 2018.

Summary

Audit Report prepared by Solidified for Taylor covering the Crowdsale smart contracts. Audit was conducted on commit 79988f5fffc5302f8dff721c9f3384b9b769203c of branch `master`.

Process and Delivery

Two (2) independent Solidified experts performed an unbiased and isolated audit of the below contracts. The debrief and cross-verification took place on January 22, 2018 and the final results are presented here.

Audited Files

The following files were covered during the audit:

* **[Crowdsale.sol](./contracts/Crowdsale.sol)**
* **[TaylorToken.sol](./contracts/TaylorToken.sol)**
* **[TaylorTokenTGE.sol](./contracts/TaylorTokenTGE.sol)**
* **[TokenVesting.sol](./contracts/TokenVesting.sol)**

Intended Behavior

The purpose of these contracts is to operate a crowdsale, distribute, and manage simple vesting of Taylor's platform token.

Note: We received an amendment to the original spec of intended behavior on 1/19 that is documented below. The audit was done on the assumptions of the original spec + amendment.

Original Spec 1/18:

https://docs.google.com/document/d/1KO9jzWWHQQ6XvqquRgM88RVUqzHZoQaegxnYoKGtzzI/edit?usp=sharing

Amendment 1/19:

The new rules for pools are:

1. Pools need to be whitelisted;
2. Pools will be able to buy up to 250 ETH during the first week. After that, the maximum amount will be the sale as the regular buyers;
3. The price will be at 0.0006 ETH during the first week. After that, they will buy at the same price as regular buyers;

4. The maximum allocated amount for pools will be 1250 ETH;

Solidified Stamp

In order to qualify for a Solidified Stamp, Taylor must address all major and minor issues reported here in the final version of the contract, make the TaylorToken ERC20 compliant and confirm the intended behaviour as described in this document.

Issues Found

1. When sale ends without the cap having been reached, the contract will never be finalized

---

Crowdsale.sol / lines 230-233

The spec does not state that the remaining tokens owned by the crowdsale contract after the sale is over should be burned, however, the code looks like it is intended to work this way. However, there is no way for the sale to be finalized without it reaching the cap specified in `tokenCap`. If the sale should end without the cap being reached (because `endTime` is reached), the `finalize()` method will never be called. It cannot be called from the outside either, since it is marked as `internal`.
This will cause the `Finalized` event to never be triggered, and the remaining tokens to never be burned.

Recommendation:

```
bool public finalized = false;

function endSale() public {
    require(finalized == false);
    require(now >= endTime);
    finalized = true;
    taylorToken.burn(taylorToken.balanceOf(this));
    Finalized(tokensSold, weiRaised);
}
```

AMENDED [2018-1-26]:

This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

2. A pool can buy up to 250 ETH also after week 0

Crowdsale.sol / lines 97-99

It is unclear if the spec means to state that a whitelisted address can only buy for <=50 ETH in total, or just per purchase, and likewise with pools and <=250 ETH. The code has checks both for per purchase (in `isValidPurchase()`) and in total (`buyTokens()`, starting line 97). For the purpose of this audit, we assume that the cap is meant to be in total.

In `buyTokens()`, starting on line 97, it does not check for which week it is. As such, it is entirely possible for a pool to buy up to 250 ETH in week 1 and beyond by simply making 5 separate purchases of 50 ETH each.

Recommendation:

We suggest having all sanity checks like this inside the `isValidPurchase()` function, and to move the checks for total investment per address there.

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

3. Crowdsale checks if non-pool contributions exceed `poolEthCap`

Crowdsale.sol / lines 104-109

In the worst case (pool contributions exceed poolEthCap before non-pool contributors purchase any), the crowdsale will only be able to raise 1250 ETH.

```
if(poolEthSold.add(amount) > poolEthCap){
    uint256 validAmount = poolEthCap.sub(poolEthSold);
    ch = amount.sub(validAmount);
    msg.sender.transfer(ch);
    amount = validAmount;
}
```

Recommendation:

Replace: `if(poolEthSold.add(amount) > poolEthCap){`

With: `if(whitelistedPools[msg.sender] && poolEthSold.add(amount) > poolEthCap){`

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

4. TaylorToken is not ERC20 compliant

Two issues prevent `TaylorToken` from full ERC20 compliance:

1. Safety measures taken in function `approve` are not backwards compatible. As delineated in the [ERC20 standard](https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md#approve), token contracts should not enforce setting allowance to 0 prior to setting a new allowance as it breaks compatibility with contracts deployed before the allowance attack was known.

2. A transfer event with the _from address set to `0x0` should be triggered when tokens are created.

Recommendation:

1. Remove the check for setting allowance to 0 prior to setting a new allowance in `approve`, and take one of the recommended approaches laid out [here](https://github.com/ethereum/EIPs/issues/738#issuecomment-336277632).

2. Add `Transfer(0x0, owner, totalSupply);` to the constructor of `TaylorToken`.

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

5. Security measures made in `approve()` are not sufficient

TaylorToken.sol / lines 186-198

The proposed security mechanism in the `approve()` method does not actually mitigate the attack (as described here: https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729)
The person making the allowance still has to check that the allowee did not manage to spend any of their allowance before the transaction setting allowance to 0 was mined.

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

6. Provide licences for third-party code

Various OpenZeppelin contracts have been included in the repo by copying rather than by package manager. This is not recommended by OpenZeppelin, though not necessarily incorrect as npm, etc. can be considered an attack vector. That said these contracts are under the MIT license, which requires its license/copyright notice to be included along with the code.

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

7. Consider moving the `burn` function

TaylorToken.sol / lines 80-90

The `burn()` function is put under the comment "owner only functions" (line 63 and on), but is is fact public (but guarded by a whitelist). Consider moving this to under the corresponding comment to avoid confusion.

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

8. Consider inheriting from well-tested libraries

Consider using more of the OpenZeppelin contracts (like BasicToken) via inheritance instead of copy-pasting and modifying. This contract is very long, and it would reduce time for future audits.

9. Consider initializing the contract in a constructor instead of `setUp()`

TaylorTokenTGE.sol / lines 38-47

The owner of the contract can call the `setUp()` function more than once, potentially changing the addresses. I don't see any reason in the spec for not having this as a constructor - especially since it is called in the deploy script.

10. Redundant code in TaylorToken

Functions `decimals` and `totalSupply` are unnecessary since state variables `decimals` and `totalSupply` are public: getter functions are automatically created for public state variables.

AMENDED [2018-1-26]:

This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

11. Remove unused variable

TaylorTokenTGE.sol / lines 12,46

The `ready` variable is not in use, and can be removed.

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

12. Consider correcting erroneous docs

TokenVesting.sol / lines 7-11

The initial comment is wrong - this TokenVesting contract is NOT optionally revocable by owner. The original OpenZeppelin contract it was copied from is. Consider fixing the comment to avoid confusion.

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

13. Extract minimum investment into a constant

Crowdsale.sol / lines 138,170

There seems to be a minimum requirement per purchase of 0.01 ETH (see line 170). This is not in the spec. Also, consider extracting it into a constant with a describing name - now it's a magic number being repeated in two places (lines 138 and 170)

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

14. Extract maximum investment for pools and normal contributors into constants

Crowdsale.sol / lines 98, 100, 173, 175

Consider pulling the magic numbers 50 and 250 (the max amount each contributor is allowed to buy if they are a pool or a normal contributor, respectively) into constants.

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

15. Remove superfluous `require`

Crowdsale.sol / lines 100, 175

Line 100:
(`require(contributors[msg.sender].add(amount) <= 50 ether`)
makes line 175:
(`require(msg.value <= 50 ether);`)
Superfluous.

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

16. Reduce scope of variable `ch`

Crowdsale.sol / lines 103-109

The `ch` variable initialized in line 103 can be moved to inside the `if` statement in lines 104:109, reducing its scope and making the code easier to read.

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

17. Reduce scope of variable `change`

Crowdsale.sol / lines 114-123

The `change` variable initialized in line 114 can be moved to inside the `if` statement in lines 116:121, reducing its scope and making the code easier to read, if line 123 is also moved to inside the `if` statement, which it safely can be. It does not do anything unless the `if` condition is true.

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

18. Always use SafeMath for subtraction

Crowdsale.sol / lines 138

In line 138, the minus operator (`-`) is used instead of `SafeMath.sub()`. Although it will probably not have any effect in this specific case, prefer using `SafeMath` for all subtraction operations.

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

19. Be explicit about which `uint` the code is using

Crowdsale.sol / lines 114, 115, 197

Consider specifying `uint256` also on the integers on line 114, 115 and 197. `uint` is an alias for `uint256`, but using the full form is preferrable. At least be consistent and use one of the forms.

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

20. Consider implementing a safety mechanism for pausing the crowdsale

There is no way to pause the presale if a bug should be discovered after the contract is deployed. Consider implementing such a feature, for instance by utilizing OpenZeppelin's `Pausable` contract.

AMENDED [2018-1-26]:
This issue has been fixed by the Taylor team and is not present in commit ab18f19404385774abb7d08d40c4541e384d9353.

Closing Summary

Several issues were found during the initial audit, which all have been addressed by the Taylor team, as noted in the amendments made.

Beyond the issues mentioned, the contracts were also checked for overflow/underflow issues, DoS, and re-entrancy vulnerabilities. None were discovered.

Audit Report for Taylor Crowdsale. January 22, 2018.

OpenZeppelin contracts such as Ownable/SafeMath/Crowdsale have been widely audited and secured, as such, they were not prioritized for auditing.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of the Taylor platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.