Security Audit of DDAO

Conclusion

In the final contract were not found:

- Backdoors for investor funds withdrawal by anyone (check critical notes).
- Bugs allowing to steal money from the contract.
- Other security problems.

Obvious errors or backdoors were not found in the contract.
The client was acknowledged about all secutiry notes below.

Scope

https://github.com/defihuntersdao-club/contract-ddao-vesting (https://github.com/defihuntersdao-club/contract-ddao-vesting)
commit: `750c7ad5`

Methodology

1. Blind audit. Try to understand the structure of the code.
2. Find info in internet.
3. Ask quiestions to developers.
4. Draw the scheme of cross-contracts interactions.
5. Write user-stories, usage cases.
6. Run static analyzers

Find problems with:

- backdoors
- bugs
- math
- potential leaking of funds
- potential locking of the contract

- validate arguments and events
- others

Result

Critical

1. Unguaranteed claiming for users.

At:

- contracts/CrowdsaleVesting.sol:62
  there is a transfer of DDAO tokens, but there is no guarantee for user, that these DDAO token will be really on the balance of the contract. So user may no be sure that he will really receive DDAO tokens.

Recommendation

Add a method to lock relevant amount of DDAO tokens on the contract address.

Client's commentary

> Tokens have already been distributed to customers and they have received aDDAO. We answer with our name. In the last change, we added token addresses directly to the contract so that it would not be visible that these are not proxy contracts. The withdrawal of tokens is needed in case we are somehow broken, despite all the tests and audits, so that we can withdraw and redo the contract.

Status

ACKNOWLEDGED

2. Owner may withdraw all DDAO tokens.

At:

- contracts/CrowdsaleVesting.sol:128

at any point of time owner may withdraw all DDAO balance, so users will not be able to claim any tokens.

Recommendation

Implement some guarantee that users will be able to withdraw their DDAO tokens.

Client's commentary

> Tokens have already been sold to customers and they have received aDDAO. We answer with our name. In the last change, we added token addresses directly to the contract so that it would not be visible that these are not proxy contracts. The withdrawal of tokens is needed in case we are somehow broken, despite all the tests and audits, so that we can withdraw and redo the contract.

Status

ACKNOWLEDGED

Major

1. Mapping is never initialized.

At:

- contracts/CrowdsaleVesting.sol:42

you use claimAddressNumber but it's never initialized.
Also here:

- contracts/CrowdsaleVesting.sol:64

it will always set nextNumber to `1`.

Recommendation

Looks like a mistake, fix it or add more comments.

Status

FIXED at b59481e7

Warning

1. Set periods in days or seconds, month is confusing.

At:

- contracts/CrowdsaleVesting.sol:34
  Usage of month may be confusing.
  Imagine, if start is set on 5th Febrary, and your period is 1 month, user may expect the the period ends on 5th March, but in fact the period ends on 7th March (because Febrary has not 30 days but 28).

Recommendation

Set periods in days or seconds.

Status

ACKNOWLEDGED

2. Confusing access duplication.

At:

- contracts/token/ERC1155/ERC1155.sol:9
  you declare the contract as Ownable, however inside ERC1155PresetMinterPauserUpgradeable there is a DEFAULT_ADMIN_ROLE and usually it is used for the same purpose as a Owner.

Recommendation

Use one Access control system, roles or ownable.

Status

FIXED at b59481e7
The contract was removed.

3. Mismatch of number in the code and in the comment.

At:

- contracts/Participants.sol:36
  the number in the code is 48351171684062000000000
  ~ 48351.171684062
  however in the comment at the same line it's 48351.171684063

Recommendation

Fix the number.

Status

FIXED at b59481e7

4. Add round verification.

At:

- contracts/CrowdsaleVesting.sol:51
  user passes the `_round` as an argument, but there is no check in the code that this round is already started.

Currently the execution will fail at contracts/CrowdsaleVesting.sol:109 with no error-message what will look like as an unexpected error for a user.

Recommendation

Check the `_round` correctness and give readable error messages.

Status

ACKNOWLEDGED

5. Dangerous potential reentry in claim.

At:

- contracts/CrowdsaleVesting.sol:60
- contracts/CrowdsaleVesting.sol:62

there are just unknown external contracts calls.
Any code can be inside addao.transferFrom and ddao.transferFrom. Maybe this is upgradeable token, maybe it let user execute selector inside. We don't know. So if external code call the same `claim` method inside, the storage will be changed incorrectly.

Recommendation

Change storage before external call or add nonReentrant protection.

Status

FIXED at b59481e7
nonReentrant was added

6. Incorrect variable name.

At:

- contracts/CrowdsaleVesting.sol:65

it's blockHash not txHash. Name txHash is confusing. You can also just use blocknumber instead of hash.

Recommendation

Rename or use just blocknumber.

Status

FIXED at b59481e7

7. Purposeless claimByAddress.

At:

- contracts/CrowdsaleVesting.sol:66

you can just emit an event to record the information about claiming.

Recommendation

Emit event.

Status

ACKNOWLEDGED

8. Centralization power of owner.

At:

- contracts/CrowdsaleVesting.sol:115
- contracts/CrowdsaleVesting.sol:119

owner may blacklist any user.

Recommendation

Introduce DAO management or window-time-period.

Status

ACKNOWLEDGED

9. Unclear calculations of available claim amount.

At:

- contracts/CrowdsaleVesting.sol

the logic behind available claim amount calculation is not really straight-forward and clear.

Recommendation

Add more comments and refactor the code to make all logic is absolutely clear from the first sight for any user.

Status

ACKNOWLEDGED

10. Infinite minting by the owner.

At:

- contracts/ERC20Base.sol:237
  owner may mint any amount of tokens at any time.
  No guarantee of dumping the token by the owner or due to leaking the owner wallet access.

Recommendation

Limit the minting by time-window or user ERC20Capped.

Status

FIXED at b59481e7
The contract was removed

Comment

1. Use openzeppelin imports instead of implement base contracts by yourself.

It seems like several contracts code were copy-pasted from Openzeppelin github.
It's better to use imports from well-known well-tested repository. It decrease the number of files in the repo and decrease the chance of typo mistake. Or if some modification is neccesary, add reference link in a comment to the original.

Recommendation

Use imports or add references.

Status

PARTLY FIXED at b59481e7

2. Use immutable and constant variables.

At:

- contracts/CrowdsaleVesting.sol:17
- contracts/CrowdsaleVesting.sol:18
- contracts/CrowdsaleVesting.sol:23
- contracts/CrowdsaleVesting.sol:24
- contracts/CrowdsaleVesting.sol:25

- contracts/CrowdsaleVesting.sol:27
- contracts/CrowdsaleVesting.sol:30
- contracts/CrowdsaleVesting.sol:31
- contracts/CrowdsaleVesting.sol:32
- contracts/ERC20Base.sol:40
- contracts/ERC20Base.sol:41
- contracts/ERC20Base.sol:42

It's more efficient to declare unchangable variables as immutable or constant. See **https://blog.soliditylang.org/2020/05/13/immutable-keyword/** (https://blog.soliditylang.org/2020/05/13/immutable-keyword/)

> The big advantage of immutables is that reading them is significantly cheaper than reading from regular state variables, since immutables will not be stored in storage, but their values will be directly inserted into the runtime code.

Recommendation

Use imports or add references.

Status

FIXED at b59481e7

3. VARIABLE IS ALWAYS ZERO.

At:

- contracts/CrowdsaleVesting.sol:28
  The variable is always zero.

Recommendation

Remove it.

Status

FIXED at b59481e7

4. PACK OPTIMIZATION FOR STRUCT.

At:

- contracts/CrowdsaleVesting.sol:36
  The struct may be efficiently packed to decrease the gas usage. See **https://medium.com/@novablitz/storing-**

structs-is-costing-you-gas-774da988895e

(https://medium.com/@novablitz/storing-structs-is-costing-you-gas-774da988895e)

Recommendation

Do struct optimization.

Status

FIXED at b59481e7

5. Use self-explaining names.

At:

- contracts/CrowdsaleVesting.sol:36-41
  it's not clear what do
  amount and number mean. Amount of what? Number of what?

Recommendation

Rename to something like ddaoTokensAmount and roundNumber.

Status

FIXED at b59481e7

6. Declare externaly used methods as external.

In a lot of methods (find the full list in the slither output below) which used externaly-only, you use `public` declarationm however with `external` declaration method calls are much cheaper.
See
https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

(https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices)

Recommendation

Use `external` declaration.

Status

ACKNOWLEDGED

7. Use `unchecked` declaration.

At:

- contracts/ERC20Base.sol:163
- contracts/ERC20Base.sol:202
- contracts/ERC20Base.sol:231
- contracts/ERC20Base.sol:278

solc starting from 0.8.0 includes overflow and underflow checks under-the-hood.
It make sense to manually disable such check because you have already underflow, check the example here:
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol#L164 (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol#L164)

Recommendation

Use `unchecked` declaration.

Status

FIXED at b59481e7
The contract was removed

8. Approve infinite amount.

At:

- contracts/ERC20Base.sol:161

there is no check for type(uint256).max (infinite amount)
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol#L162 (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol#L162)

Recommendation

Support approving of infinite amount.

Status

FIXED at b59481e7
The contract was removed

9. Burn instead of transfer.

At:

- contracts/CrowdsaleVesting.sol:60

it looks like here it could be just burnFrom instead of transferFrom.

Recommendation

Think about using of burnFrom.

Status

ACKNOWLEDGED

10. Optimize the code.

At:

- contracts/CrowdsaleVesting.sol:110

```
uint256 secondsPassed = _timestamp - (startDate + lockupPeriod);
secondsPassed = secondsPassed > _vestingPeriod * oneMonth ? (_vestingPeriod * one
```

means

```
uint256 secondsPassed = _timestamp - (startDate + lockupPeriod);
if(secondsPassed > _vestingPeriod * oneMonth) {
    secondsPassed = _vestingPeriod * oneMonth;
} else {
    secondsPassed = secondsPassed;
}
```

so in else-branch you just set the variable to itself.

Recommendation

Rewrite it:

```
uint256 secondsPassed = _timestamp - (startDate + lockupPeriod);
if(secondsPassed > _vestingPeriod * oneMonth) {
    secondsPassed = _vestingPeriod * oneMonth;
}
```

Status

ACKNOWLEDGED

11. Avoid double calculation.

At:

- contracts/CrowdsaleVesting.sol:74-75

you call the same `calculateUnlockedTokens` function twice.

Recommendation

Calculate the value only once.

Status

FIXED at b59481e7

12. DECLARE RESULT VARIABLE IN THE FUNCTION DEFINITION.

At:

- contracts/CrowdsaleVesting.sol:82
- contracts/CrowdsaleVesting.sol:132

you delcare the variable inside the function body and spend gas, but memory storage for the result is already allocated and you can use it if you declare the function in such way:

```
function balanceOf(address _address) public view returns (uint256 result) {
```

Recommendation

Use such declaration:

```
function balanceOf(address _address) public view returns (uint256 result) {
```

Status

ACKNOWLEDGED

Slither static-analyzer log

https://pastebin.com/mV39CAm6 (https://pastebin.com/mV39CAm6)