SMART CONTRACTS EXPERTISE — RIGHT WAY TO SUCCESS
CryptoBnB Smart Contract Audit

If you have any questions concerning smart contract design and audit, feel free to contact zoia@oceanico.io

Contents

Description of the complex of procedures for auditing a smart contract

1. Primary architecture review

- Checking the architecture of the contract.
- Correctness of the code.
- Check for linearity, shortness and self-documentation.
- Static verification and code analysis for validity and the presence of syntactic errors.

2. Comparison of requirements and implementation

- Checking the code of the smart contract for compliance with the requirements of the customer code logic, writing algorithms, matching the initial constant values.
- Identification of potential vulnerabilities.

3. Testing according to the requirements

- Control testing of a smart contract for compliance with specified customer requirements.
- Running tests of the properties of the smart contract in test net.

Terms of Reference for the creation of a smart contract

ICO Crowdfunding Details:

Token Specs:

- Total CKey Tokens 1 billion
- 1 CKey=$ 0.08
- Token Sale to run for 90 days
- HardCap is $15.2 Million
- Minimum Cap $1.5Million
- Minimum Purchase is $75 (0.1 ETH)
- CKey token is ERC20
- Participants wallets must be ETH ERC20 compatible
- Tokens are bound to USD for the sale, as ETH fluctuates the amount of tokens received changes not the price of the tokens.

Token Dispatch and Listing

- Participants will register on the Cryptobnb.io (no other website or mechanism is promoted).
- After filling out the KYC form, the CryptoBnB team will vet all applicants. Team will notify approval or rejection with ETH wallet address
- Upon successful ETH transfer a notification will be sent to applicant.
- Allocated tokens will be transferred to applicant's wallet in maximum of one week.
- Tokens will be listed in major exchanges within 1–2 months from the tokens dispatch date.

Token Distribution Tier Model:

	Tokens	Disc	Token Price
ICO T1	100,000,000	50%	$0.0400
ICO T2	80,000,000	40%	$0.0480
ICO T3	60,000,000	30%	$0.0560
ICO T4	40,000,000	20%	$0.0640
ICO T5	20,000,000	10%	$0.0720
Final Price		100,000,000	$0.08
Total	**300,000,000**	**100,000,000**	

Token Distribution:

Token Allocation	
Reserve	22.50%
Token ICO	30%
Team	7.50%
Founders	20%
Advisors	2.50%
Bonuses Host	10.00%
Bonuses SP	5%
Bounties	2.50%
Sum	**100%**

Data needed:

Interested users provide their names, IDs, KYC info, email, and Ethereum address.

Smart Contracts Process:

- Users register through KYC form in the cryptobnb.io website
- CryptoBnB Inc. will vet all applicants and award Tokens accordingly. Rejected applicants will be notified.
- Users will be given the wallet address to send their ETH
- A smart contract will be generated creating 1 billion CKey

- The smart contract will have various pools/tranches for the Tokens as per Tokens allocation plan
- Wallets of participants/teams/ founders will be allocated in the appropriate pool
- Smart contracts will be released to Mainnet
- Held u token will be left in escrow accounts
- Excess Tokens from each pool will be swept after ICO to a freezer account.
- 1% of frozen Tokens will be donated to an accredited cancer association done manually
- CryptoBnB Inc. reserve the right to move tokens from one tier to another as they deem needed.
- HardCap is set at $15 Million
- Minimum viable funds will be set at $5 Million, if the amount is not reached the funds will be returned.
- Minimum purchase is 0.1 Ether.
- In the unlikely event of a total refund, gas to refund will be paid by the participant , the ether sent by them will become available for refund from the multisig wallet
- If there is a remainder of ethereum ETH left (dust) it will not be worth the gas to send back, the crowdfunding will absorb it later.
- Shareholder and Founders will be able to trade on exchanges up to 50% of their Tokens immediately, the remaining will be frozen for 12 months. Team Members will have their Tokens frozen for 12 months.
- Reserve, Bonuses and Bounties can be traded in the exchange upon listing

List of audited files

Github of the project

https://github.com/Cryptobnb/Smartcontracts

The following 10 .sol files were the object of the audit:

- BasicToken.sol
- CBNBCrowdSale.sol
- CBNBTeamWallet.sol
- CBNBToken.sol
- ERC20.sol
- ERC20Basic.sol
- Migrations.sol
- Ownable.sol
- SafeMath.sol
- StandardToken.sol

Review of smart contract #1

CryptoBnB token review#1.
CBNBCrowdSale.sol

https://github.com/Cryptobnb/Smartcontracts

Important:

1. Solidity 0.4.18 has to be used. This is the latest stable version with security fixes.

2. The modifier icoHasEnded checks both conditions: the cap is reached and time is out. It has to use the ‘or’ operator.

3. The method RefundWithdrawal uses the modifier icoHasEnded. It checks if the cap is reached. The next modifier is refundOnly that checks that minLimit is not reached, so investors with the status New or Approved can never get a refund. Investors with the status Denied can get a refund only if the cap reached and time is out.

4. If the investor invests twice or more with the same address, its status resets to New each time.

5. ValidPurchase checks that the cap is less or equal to weiRaised plus the transaction value. It means that the last transaction volume should be exactly equal to the cap minus weiRaised, in order to: 1) validPurchase to allow it 2) the cap to be reached. The correct approach is to accept the transaction and return the delivery to the investor if the transaction value plus weiRaised is greater than the cap.

6. The method buyTokens transfer all eth to multiSigWallet, so RefundWithdrawal cannot return eth.

Code quality:

1. Check now >= icoStartTime makes no sense because the constructor set icoStartTime to now.

2. The modifier ContractPaused checks that the contract is not paused. It shoud be renamed ContractActive.

3. SaleTier.StartTime is never used, onTheWhitelist is never used.

4. 5*10**17 should be written as “0.5 ether” and switched to constant. 10**10 should be switched to constant. 42 can be written as TIER_DURATION*TIER_COUNT, both constants should be specified.

5. saleTier initialization should be written as a cycle. It reduces the chance of human error in the code and makes the code shorter.

saleTier[i].price should be calculated as (3900 - 150 * i) * 10**18.

salesTier[i].endTime should be calculated as now + 7*i days.

Attention: 3900 is ‘token per eth’ value, while now price is ‘eth per token’. It looks more clearly when‘token per eth’ is used every time. In this case, the following improvements will be required: the calculation qtyOfTokensRequested, buyTokensRemainingWei should use mult. The calculation remainingWei should use div.

6. weiRaised has no initialization in the constructor.

7. The constructor has no check _teamAddress argument set to zero.

8. Convert zero to address when there is a check that an address is not zero.

9. The methods cleanup and finalize are both onlyOwner, and cleanup is internal, so this modifier can be removed from cleanup.

10. The methods transferUnsoldICOTokens do not transfer anything, they only calculate the amount of unsold tokens. It will be more appropriate to rename it getCountUnsoldICOTokens. Also, 6 should be a constant.

Review of smart contract #2

CryptoBNB token review #2

https://github.com/Cryptobnb/Smartcontracts

Important:

1. The method buyTokens has a modifier icoHasEnded, so it can never be executed. It is highly recommended to create a modifier icoActive, that checks that weiRaised < cap && and so on, and to use it with buyTokens.

2. The method buyTokens under some conditions returns delivery to

   multiSigWallet.transfer(msg.value);;

   And sends msg.value to wallet:

   multiSigWallet.transfer(msg.value);

   If (weiRaised.sub(cap) > 0), it will lead to a reversal, because the contract never stores ether.

It is recommended to calc delivery for investor at the beginning of this method, if msg.value+weiRaised > cap, and then work with msg.value.sub(delivery).

If the last token is sold, just add the remaining weis to the delivery, and transfer the delivery to the investor, if it is greater than zero. In this way, no manual refund will be needed.

3. The method buyTokens does not check the case when an investor tranfers such a large amount that it covers not only the current tier, but one or more of the subsequent tiers too. This is a hypothetical situation, but it is better to foresee it. A good solution is to use a cycle here.

Code quality:

1. The following variables are never used: lockTier, decimals.

2. It is good to create a constant ETH_DECIMALS = 10**18.

3. In the following line the comment ‘wei’ has to be

   saleTier[i].price = (3900 - 150*i); //tokens per eth

4. The modifier contractPaused was not changed after the first review. It is recommended to rename it contractActive because it checks that paused == false.

5. The following code is not needed, because enum in Solidity is uint8 internally, so Status.New

== 0, and‘default value’ for any variable are always zeros:

```
if(investors[msg.sender].whitelistStatus != Status.Approved){

investors[msg.sender].whitelistStatus = Status.New;

}
```

6. In the method buyTokens, use TIER_COUNT here: if (tier <= 5){

7. cap.sub(weiRaised); does nothing. There is no assignment.

Review of smart contract #3

CryptoBNB token review #3

https://github.com/Cryptobnb/Smartcontracts

Important:

1. If an already whitelisted investor buys tokens for the second time, the tokens will not be automatically sent to the investor. So the owner will need to add this investor to the white list one more time.

2. The modifier icoHasEnded was broken with the last commit, it's recommended to roll it back, remove this modifier from buyTokens, create a modifier icoActive:

   ```
   require(weiRaised < cap && now < icoEndTime && calculateUnsoldICOTokens() > 0);
   ```

   and use this modifier in the buyTokens method.

3. There are bugs in the calculation tokensSold during the transition to the next tier. At first,

   ```
   142        qtyOfTokensRequested = tierRemainingTokens;
   ```

   where tierRemainingTokens is the remainder from the previous tier. Next,

   ```
   148        qtyOfTokensRequested += buyTokensRemainingWei;
   ```

   where buyTokensRemainingWei is tokens for a new tier. Finally,

   ```
   160        saleTier[tier].tokensSold += qtyOfTokensRequested;
   ```

   so the tokensSold value in the new tier will be the sum of tokens for both tiers.

4. A problem in the lines 162..169. If an investor is whitelisted, contrAmount will not be increased. So the investor will not be able to get a refund. In fact, contrAmount has to be increased regardless of the whitelist status.

5. Refund is impossible due to a reversal in the function ().

Review of smart contract #4

Cryptobnb smart contract review #4

https://github.com/Cryptobnb/Smartcontracts

Important:

1. Solidity 0.4.19 is released. It's recommended to update the compiler and change the pragma.

2. Tokens are transferable during the ICO. It's recommended to add a 'paused' variable to CBNBToken and override the methods transfer and transferFrom to check !paused || msg.sender==crowdsaleContract

3. It's recommended to use a git submodule to include zeppelin-solidity to repository. In the directory cryptobnb/contracts/ (the directory containing CBNBToken.sol) run the following command:

   $ git submodule add https://github.com/OpenZeppelin/zeppelin-solidity

   This command will create a zeppelin-solidity directory. Do not add it to git with 'git add'. Next, add a dependency to repo:

   $ git add ../.gitmodules

   Change imports in CBNBToken.sol, CBNBCrowdsale.sol, CBNBTeamWallet.sol

   import "./zeppelin-solidity/contracts/token/StandardToken.sol" and so on.

   After push and pull/clone, the dependencies can be updated on another computer:

   $ git submodule init && git submodule update.

   This will enable an actual zeppelin-solidity that removes compiler warnings. All files but CBNB*.sol can be removed from the contracts directory.

4. The method transferTeamTokens in CBNBTeamWallet.sol uses the bnbToken.transfer- From(this, msg.sender, sendValue) method. It doesn't work, so it's recommended to use bnbToken.transfer(msg.sender, sendValue).

   use bnbToken.transfer(msg.sender, sendValue).

Review of smart contract #5

Cryptobnb smart contract review #5

https://github.com/Cryptobnb/Smartcontracts/

Important:

1. When purchasing tokens, the owner of the Crowdsale contract usually purchases tokens in which tokens are not actually credited to anyone, and the totalTokensSold counter is incremented. In my opinion, this option is not necessary. I recommend adding a check to the top of the method require(msg.sender != owner).

2. The buyTokens function contains errors. For example, in line 167, instead of mul(price), there must be a div (price).

```
167        buyTokensRemainingWei = (remainingWei.mul(price)).mul(TOKEN_DECIMALS);
```

3. The price of the token is not recalculated when purchasing tokens from several Tier.

4. Due to the appearance of SaleTier storage tiers = saleTier[tier]; a new bug appears. Now tiers (highlighted in the screenshot) is not a saleTier[tier], but saleTier[tier - 1] because of tier++ in line 164. tiers did not point to saleTiers[tier++].

```
161    if (qtyOfTokensRequested >= tierRemainingTokens){
162      remainingWei = msg.value.sub((tierRemainingTokens.div(TOKEN_DECIMALS)).mul(price));
163      qtyOfTokensRequested = tierRemainingTokens;
164      tier++;
165
166      if (tier < TIER_COUNT){
167        buyTokensRemainingWei = (remainingWei.mul(price)).mul(TOKEN_DECIMALS);
168        qtyOfTokensRequested += buyTokensRemainingWei;
169        tiers.tokensSold += buyTokensRemainingWei;
170        remainingWei = 0;
171      } else {
172        msg.sender.transfer(remainingWei);
173      }
```

5. There remains another problem with the calculation of tiers.tokenSold. If you purchased tokens of several Tier, you need to fill out the saleTiers[tier].tokensSold before tokensToBeSold, saleTiers[tier + 1].tokenSold and so on.

Code quality

1. Miscellaneous indents (indents, getEtherPrice, pausedContract).

2. The brackets do not play any role in the calculation order (not only in this place, there are many such examples in the whole project). They only make the code cluttered. Recommend to remove.

```
166     if (tier < TIER_COUNT){
167       buyTokensRemainingWei = (remainingWei.mul(price)).mul(TOKEN_DECIMALS);
168       qtyOfTokensRequested += buyTokensRemainingWei;
```

3. It is customary to write function names lowercase.

```
272   function RefundWithdrawal()
273     external
274     pausedContract
275     icoHasEnded
276     returns (bool success)
```

4. decimals in the CBNBCrowdSale contract is initialized, but is not used anywhere. decimals are needed in CBNBToken to implement ERC20, and it is not needed in the crowdsale contract if it is not used anywhere in the code.

5. The purpose of the getTokenPrice method is not defined. Its functions are duplicated by the getEtherPrice method.

Testing in testnet

Test scenario:

1. Creating contracts

2. Simple purchase of several tokens. Check that you bought as many tokens as you expected.

3. Buy tokens from Tier2. Check that the price of the token has changed automatically.

4. Buying tokens is more than what is left to buy until the end of the stage (tier). Checking that the tokens are credited, as expected.

5. Checking that finalize can not be called until the ICO is completed

6. Checking that finalize is possible to trigger after the sale of all the tokens

7. Prohibit the purchase of tokens. Check that the refund is working.

Tests performed:

1. **Success**
   Deploying CBNToken, CBNBTeamWallet, CBNBCrowdSale in the Ropsten test network.

   Executing CBNToken.setCrowdsaleContract (CBNBCrowdSale.address), CBNToken.approve (CBNBCrowdSale.address, 100000000000000000000). All transactions passed without errors.

2. **Failure**
   Purchase of buyTokens tokens by the owner of contracts for 0.2 ETH.
   The transaction was successful, the totalTokensSold counter increased by 20'000'000'0000000000. This behavior is undesirable (see Important.1)

3. **Success**
   Purchase buyTokens tokens from another address at 0.2 ETH.
   The transaction was successful, the totalTokensSold counter increased by 20'000'000'0000000000. All as expected.

4. **Success**
   Running whitelistAddresses (adddress, true). 20'000'000 reserved tokens were transferred to the specified address.

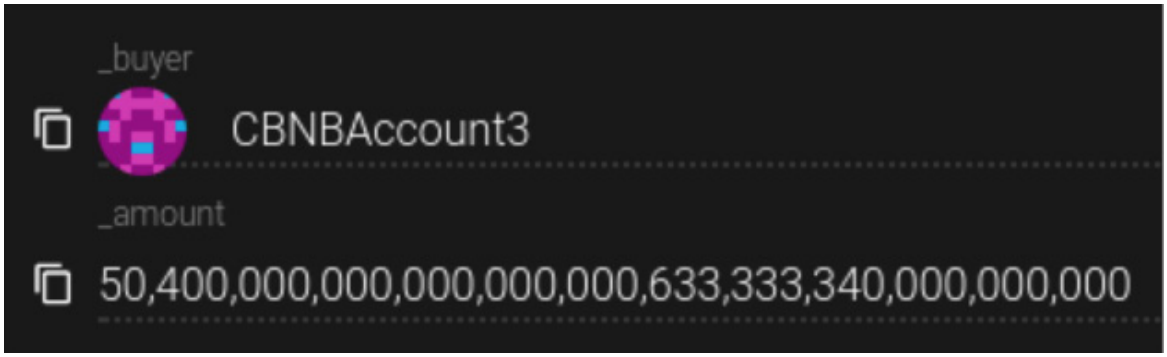
5. **Success**
   Purchase of remaining (in Tier1) 60'000'000 tokens. Sent by 0.6 ETH. The balance of tokens replenished by 60'000'000 CKey.

6. **Success**
   Check for automatic change in the price of tokens in Tier2.
   Purchase of tokens at 0.2 ETH. Credited 16'666'666 CKey.
   The test showed that the token has a new price of $ 0.048.

7. **Failure**
   Calculations show that, in order to buy the remaining 63'333'334 CKey in Tier2, you need to pay 0.760000008 ETH. In Tier3, the CKey price of the token should be $ 0.056.

   Let's try to buy 63'333'334 + 30'000'000 CKey for 0.760000008 + 0.42 ETH.
   As a result, an extremely large number of tokens is reserved, the number of which is accustomed to the maximum issue pledged in the contract.

   Error in calculations. You can not continue testing (see Important.2-5)



As a result of the verification and testing of the smart contract in the Ropsten test network, the following was identified:

- a number of insignificant comments related to the quality of the code does not affect the work.

- critical vulnerability. This vulnerability is highly unlikely under normal tokenail conditions (see Important.2-5), does not affect the security of investor funds placed on a smart contract, but in the event of an attack it can disrupt further tokenization.

Review of smart contract #6

Cryptobnb smart contract review #6

https://github.com/Cryptobnb/Smartcontracts/

Important:

1. You should pass a block.timestamp or now instead of a block.number to the setFreezeTime function and you should replace teamWallet in finalize with bnbTeamWallet and remove unused teamWallet.

Code quality

There are no comments or recommendations on the quality of the code.

Testing in testnet

Test scenario:

1. Creating contracts
2. Simple purchase of several tokens. Check that you bought as many tokens as you expected.
3. Buy tokens from Tier2. Check that the price of the token has changed automatically.
4. Buying tokens is more than what is left to buy until the end of the stage (tier). Checking that the tokens are credited, as expected.
5. Checking that finalize can not be called until the ICO is completed
6. Checking that finalize is possible to trigger after the sale of all the tokens
7. Prohibit the purchase of tokens. Check that the refund is working.

Tests performed:

1. **Success**
   Deploying CBNToken, CBNBTeamWallet, CBNBCrowdSale in the Ropsten test network.

   Executing CBNToken.setCrowdsaleContract (CBNBCrowdSale.address),

   CBNToken.approve (CBNBCrowdSale.address, 100000000000000000000).

   All transactions passed without errors.

2. **Success**
   Purchase of buyTokens tokens by the owner of contracts for 0.2 ETH. The transaction was successful.

3. **Success**
   Purchase buyTokens tokens from another address at 0.2 ETH.

   The transaction was successful, the totalTokensSold counter increased by 20'000'000'0000000000. All as expected.

4. **Success**
   Running whitelistAddresses (adddress, true). 20'000'000 reserved tokens were transferred to the specified address.

5. **Success**
   Purchase of remaining (in Tier1) 60'000'000 tokens. Sent by 0.6 ETH. The balance of tokens replenished by 60'000'000 CKey.

6. **Success**
   Check for automatic change in the price of tokens in Tier2.

   Purchase of tokens at 0.2 ETH. Accrued 16'666'666 CKey.

   The test showed that the token has a new price of $ 0.048.

7. **Success** Buying tokens is prohibited. Refund works.

As a result of the verification and testing of the smart contract in the Ropsten test network, it was revealed:

- an error that does not allow to call the Finalize function responsible for the translation of TEAM tokens. If you do not remove it, the TEAM tokens will be blocked.

This error does not affect the safety of funds of investors placed under an intellectual contract.

Review of smart contract #7

Cryptobnb smart contract review #7

https://github.com/Cryptobnb/Smartcontracts/

Important:

1. After the transferTeamTokens function is added, the situation is now possible: the owner calls the transferTeamTokens method before the completion of the ICO, the tokens are added to the team's wallet, but are not frozen (the freeze occurs during finalization) and team members can withdraw funds from the wallet.

Comments

1. Checking minLimit and cap. minLimit = 1500 ETH. 1500 ETH at the current ether price = $ 1.7 Million (and should be $ 1.5 Million). To increase the accuracy of minLimit, you must pass the current ETH price to the constructor. But before the end of the ICO course can change a lot, and then the minLimit will be too small or too big and the organizers of the ICO will not raise enough funds. The same goes for checking the cap.

2. LogWithdrawal (msg.sender, this.balance); // do we really want to broadcast the tehis? - Yes, it is necessary to log all operations related to transfer of ETH and tokens.

Testing in testnet

We tested the possibility of withdrawing funds from teamWallet before calling finalize (see Important.1)

1. De-contracting, setting the price of ETH, running token.setCrowdsaleContract, teamWallet.setCrowdsaleContract, token.approve.
2. crowdsale.transferTeamTokens
3. token.activate
4. teamWallet.addMember (address, 1'000'000'0000000000)
5. teamWallet.transferTeamWallet (from address)

As a result of the verification and testing of the smart contract in the Ropsten test network, it was revealed:

- Checking the token balance shows that the tokens were transferred to the registered team member, although finalize has not yet been executed. Tokens were not frozen.

This error does not affect the safety of funds of investors placed under an intellectual contract.

Results of contract audit

https://github.com/Cryptobnb/Smartcontracts

The information in this report is a list of tips and recommendations on what to look for and what needs to be done to ensure the performance of a smart contract.

OCEANICO experts conducted a smart contract audit in seven iterations. Based on the results of each iteration, the developer's developers were given recommendations for eliminating errors committed while writing the code and possible vulnerabilities of the contract.

During seven complete test run runs in the Ethereum test grid, all the errors found were eliminated by the developer. This smart contract corresponds to the specifications stated in the terms of reference and does not contain previously identified code and vulnerabilities errors.

If changes are made to the functionality of the contract, please submit the smart contract for re-examination to the OCEANICO experts (zoia@oceanico.io).