SkinCoin Code Review

rev 2

1. INTRODUCTION

This code review is our best effort to review the provided source code of the SkinCoin smartcontracts. The ac33d8e4c30e0b99e80dc202198d0fdeb6d5773e version and the following contracts have been reviewed:

```
ERC20Basic.sol
Ownable.sol
Pausable.sol
ERC20.sol
BasicToken.sol
SafeMath.sol
Crowdsale.sol
Migrations.sol
PullPayment.sol
SkinCoin.sol
StandardToken.sol
TokenSpender.sol
```

Since it is a source code security audit, only security considerations have been annotated. These other aspects are not reviewed here:

- Contracts functional design
- Design, deployment, operation and contingency plan of the ICO
- EVM code generated
- External wallet used

Critically analyse the provided information: code reviews are hard, and some of the analysis provided in this code review could be erroneous or coming from a misinterpretation of the operations done in this SmartContract.

This review is an opportunity to re-read your code: Instead of checking only the points proposed, consider re-reading all the code to find additional causes of trouble.

A Bug Bounty is a must: Since code reviews are not enough to ensure the SmartContract safety, a public bug bounty is essential, also gives good feelings about the overall security of the ICO process. An good example is the Aragon Bug Bounty Program.

2. REVIEW RESULTS

2.1 Severe

2.1.1 Call to an untrusted external

In the `approveAndCall` function no check is done to verify that the supplied parameter is the deployed CrowdSale, so any contract could be called using the SkinCoin contract as a msg.sender. This is a source of potential recursive call attacks and unauthorized interactions of the coin contract with other existing contracts.

We also recommend not to add specific methods to the Token to handle its own ICO, since this code is not going to be used further.

2.1.2 coinPerEth variable cannot be modified

In L83 specifies that

```
coinPerEther = 10000;      // will be update every 10min based on the kraken
ETHBTC
```

But there are no callable functions that modify this variable, so this variable will never be able to be modified

2.2 Potential problems

2.2.1 Possible loss of ownership

As `transferOwnership` does not check if the new owner is zero, losing ownership by error could cause that the `drain()`function stops to be failsafe. Ownable.sol transferOwnership() should check if the address is zero.

2.2.2 Unit tests are not covering important checks

Some important checks in the code that are not covered by unit tests, consider to add tests to cover them:

- minCapNotReached throw in L61
- respectTimeFrame throw in L66
- throw in L90
- throw in L109
- using a non-bonused amount in L131
- throw in L143
- Events generated

Since it is important to cover all functions, we recommend to add an additional trivial check also for the `drain()` function.

2.2.3 Make backers data externally accessible

In case of troubleshooting it will be interesting to know the contents of the backers map. Consider adding a getBacker(…) function to retrieve that information.

2.2.4 Owner can corrupt the ICO by mistake

The `start()` function can be called by mistake at the end of the ICO, thus causing an unexpected behaviour. Consider allowing to start the ICO only if `startTime` is zero.

2.2.5 Fallback emergency stop

There is no emergency stop for the fallback function (that uses complex logic calling receiveETH function), so consider using also the `stopInEmergency` modifier in the fallback function.

2.3 Warnings

2.3.1 Use safemath everywhere

Check the usage of safemath, for instance `asyncSend` is not using safemath.

2.3.2 Unit tests should be atomic

It is not a good practice to make unit tests dependent on each other; unit tests should be independent from one another.

2.3.3 Code duplication

Some of the contract seem to come from OpenZeppelin contracts. We strongly recommend using the exact OpenZeppelin contracts without removing anything, because they have been already reviewed and well tested.

In case the code needs to be modified we recommend to:

- Specify the hash of the commit of the original file
- Explicitly document the modifications over the original source code
- Adjust the test set for the modified file

Specifically, the `transfer`, `transferFrom` and `approve` functions are defined returning a boolean but they currently throw an exception if transfer conditions are not met. This could be avoided with:

- Added comments explaining why this mismatch between specification and implementation exists; or
- Implementation matching specification

2.3.4 Multisig destination limitations

Check that the destination multisig fallback function does not use more than 21000 gas. Since it is using the send function, send will fail in multisig if fallback function gas

consumes more than the prefixed 21000 gas. Review the multisig wallet and consider using `.call.value()`-based transfers.

2.4 Minor issues & comments

2.4.1 crowdsaleClosed as a function

As `crowdsaleClosed` could be calculated from the existing variables, consider creating a function called `isCrowdsaleClosed` using the existing variables.

2.4.2 Modifiers usage

Date checks are done within the functions and also using the `respectTimeFrame` modifier. Consider always using modifiers for date checks or just do not use them at all.

2.4.3 Use standard solidity documentation tags

Consider using the specific solidity tags like `@title` and `@dev`, `@param` and `@return` in documentation.

See http://solidity.readthedocs.io/en/latest/layout-of-source-files.html?highlight=natspec#comments

2.4.4 Code style

Non-normalized code should be considered harmful because code readers get the impression that code has not been double-checked before its release.

Consider fixing the following:

- **Bad indentations**: for instance HERE or HERE
- **Underscores in parameters**: for instance HERE
- **Review names**: for instance HERE `event Logs` is too generic and is only used once (take note that prefixing all events with Log is nice). It will make more sense using a name like `event CoinsEmited`. Another example is HERE, where `coinSentToEther` could be better understood naming it as `coinSentToContributors`
- Large integers: for instance HERE, `totalSupply = 1000000000000000;` can be rewritten as `totalSupply = (10 ** 9) * (10 ** decimals)`
- **Consistent spacing**: for instance HERE and HERE there's a different criteria for spacing between functions. Extra spaces HERE is also an example.
- **Document all functions and parameters**: for instance HERE there's no documentation about this function
- **Magic numbers**: consider not using magic numbers like HERE or HERE and define them as contract constants
- **Always define the access of variable and methods**: for instance, it is neither defined HERE nor HERE

2.4.5 Unit tests contain misspelled words

For instance, HERE it is using `Craudsale` instead of `Crowdsale`

2.4.6 Wrong documentation for running the tests

Tests should be run with a testrpc with a large amount of ethers. We used the following configuration:

```
testrpc

  --account="0xbaaec80bc2e3a141a9835a46856a90041785cc7339b0d532a57f5b0947d9aca0,1000000000000000000000000"

  --account="0xa9adb8aa0cdf4a90068f3e4d8cb9dd96e5099e5e0b83f9ccffbe195bd156d5e9,1000000000000000000000000"

  --account="0x61336d287b4c671047dd495007af6bce2b4925dfb6506b761f4a903d9feff602,1000000000000000000000000"
```

2.4.7 Consider using modern style test scripts

OpenZeppelin style tests scripts are easier to understand; thus, so they can be written, maintained and reviewed in an easier way.

2.4.8 Consider checking the ETH balance in refunding test

In the `Refund the payments {from: buyer}` test, no checks verifying the ether account balance are done. We recommend doing it.

2.4.9 Unused variables

`maxCapReached` is not used anywhere in the code. Consider removing it.

2.4.10 Using block numbers instead block dates

In general is safer to use block numbers instead block timestamps because miners can perform some manipulation over them. Block numbers then can be aproximated with expected block heights and amounts. Nonetheless, the potential damage is not considered harmful in this contract.

4. CONCLUSION

Reviewed smartcontracts show the expected common security practices for safe development and contains simple logic to avoid complex scenarios and future problems. Nonetheless, due to the two severe and 5 potential problems found during the review, we recommend fixing them and proceeding to a public bug bounty program, as usual in public ICOs.