V1.0

FansUnite

Audit

Dean Eigenmann
October 22, 2017

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bugfree status. The audit documentation is for discussion purposes only.

Overview

The audit was performed on all smart contracts found the FansUnite repository (https://github.com/FansUnite12/FAN-contracts), the contracts are found in the directory: contracts. The commit hash this audit was performed upon is: 1c989dbd269958df1d474d5e29ed11e96e7efee0

The following contracts were ignored in this project. This is because they are assumed to be secure as they stem from previously audited projects:

- Ownable.sol
- SafeMath.sol

Audit Results

Generally speaking the code style is good and various tests have been supplied. There are minor suggestions to improve the legibility as well as the functionality of the code.

Crowdsale tests fail and require improvements.

Severe issues were found that are in urgent need of a fix.

FansUniteCrowdfund.sol

The FansUniteCrowdfund contract, implements the token generation logic during the sale as well as the extra tokens generated post sale. External interaction is limited to ether transactions as well as the usage of the mint function on the token contract.

The contract inherits from the Ownable contract and uses SafeMath for uints.

Suggestions

1. MinGoalReached event can be removed as it is never used.
2. Considering the nature of the way the sale works, all tokens could be preminted and transferred to the crowdsale. Eliminating the need of a mint function.

Modifiers

onlyDuringSale

Ensures that the current time is greater than or equal to the address specific start time and that it is smaller than end time.

tokenCapNotReached

Ensures that the tokens sold is less than the ico supply.

onlyWhenFinalized

Requires that the finalized variable is true.

onlyWhenEnded

Requires current time to be past the end time or the entire token supply to be sold.

Constructor

- Sets the FansUniteToken.
- Sets the various addresses where tokens will be transferred to.
- Sets the various token supplies, including those being vested and those available during the ICO.
- Sets the start times for whitelist addresses as well as normal sale in addition to the token sale end time.

No issues detected in the constructor. Minor changes could be made however, from line 80-83 the multiplier 10**24 is used repeatedly. A constant variable called multiplier could be created in place.

Default Function

The default function simply forwards calls to the doPurchase function.

doPurchase

The function is responsible for the logic of token sale, this includes minting as well as accounting for the sold tokens.

The contract ensures that the sale has started and that the cap has not been reached.
Additionally it validates that more than 0 ether has been sent.
The tokens that a user will receive is then calculated by the sent ether amount multiplied by the rate provided by the getRate function.
The amount of ether received is added to the weiRaised variable, and the token amount is added to the tokensSold variable.
Tokens are minted using the mint function provided by the token.
Ethereum is transferred to multisig.

Severe

Although using the modifier tokenCapNotReached, the function does not respect the icoSupply variable. Meaning an amount of ether could be sent which would exceed the icoSupply. This would allow for more tokens to be sold through this function than are really allowed. If this were to happen, the finalize function would fail.

This bug is very likely to occur through a simple transaction that had no malicious intent. As it is common for buyers to transfer more ether than still allowed.

Suggestions

1. Check that tokensSold + tokens does not exceed icoSupply
2. Token mint function should use the _owner rather than msg.sender.
3. Use weiAmount instead of msg.value in transfer function

finalize

The finalize function is responsible for performing various allocations.

Suggestions

1. Add a check to ensure that we are not minting 0 tokens to the unsold token address.

addToWhitelist

Adds provided address into the whitelist mapping with the value of true.

Suggestions

1. Add function so an array of addresses can be passed.

startTime

Returns the start time for a specified address. If an address has been whitelisted a different start time will be returned than usual.

addPrecommitment

Function ensures that now is before the sale start and ensures that the balance provided as a function argument is greater than 0.
It adds the provided balance to the token sold variable.
Finally tokens are minted to the address that is provided.

Severe

Although using the modifier tokenCapNotReached, the function does not respect the icoSupply variable. Meaning a balance could be supplied which would exceed the icoSupply. This would allow for more tokens to be sold through this function than are really allowed. If this were to happen, the finalize function would fail.

addPrecommitmentBalances

This function iterates an array of addresses and balances calling the addPrecommitment functions for each of the entries.

getRate

Returns a specific rate depending on the amount of weeks that have passed since the official sale start time.

tokenTransferOwnership

This function transfers the ownership of the token to a specified address once the sale has been finalized.

Suggestions

1. Add a separate minter address to the token, this wouldn’t require a transfer of the ownership but rather a minter. This could be done within the token itself rather than proxied through the crowdsale.

FansUniteToken.sol

This token implements the basic token logic, it inherits the Ownable contract and implements the ERC20 interface provided by open zeppelin.

Suggestions

1. Considering Open Zeppelin is already in use, the full token implementation could be used from there.

Modifiers

canMint

Ensures that the totalSupply does not exceed the maxSupply.

Constructor

- Sets name
- Sets symbol
- Sets maxSupply from provided parameter
- Sets totalSupply to default value of 0.

transfer

The transfer function implementation follows past implementations as those provided in Open Zeppelin. No issues could be detected.

transferFrom

The transferFrom function implementation follows past implementations as those provided in Open Zeppelin. No issues could be detected.

mint

The mint function implementation follows past implementations as those provided in Open Zeppelin, however a check is executed to ensure the totalSupply after minting does not exceed the maxSupply. No issues could be detected.

TokenVesting.sol

This is contains the vesting logic, it is based on the Open Zeppelin implementation, however it does not include the cliff and revocable functionality.

Constructor

- Sets beneficiary
- Sets vesting start time
- Sets vesting duration

Suggestions

1. Vesting duration could be specified in weeks through the constructor, by multiplying by the weeks global.

release

This function is identical to the one found in open zeppelin, no issues were detected.

releaseableAmount

Returns the vestedAmount subtracted by the amount of tokens that has already been released

vestedAmount

Returns the amount of tokens that have been vested for a time period. If the duration is exceeded the total vested amount is returned. Otherwise a calculation is executed, this is identical to the one found in open zeppelin.