Fiinu Token Audit

OPENZEPPELIN SECURITY | OCTOBER 30, 2017

Security Audits

The Fiinu team asked us to review and audit their Fiinu Token (FNU) and crowdsale contracts. We looked at the code and now publish our results.

The audited contracts are located in the fiinu/smart-contract repository. The version used for this report is the commit `7589b9d137bbf902e142e4f0476b06976e390bdb`.

Here's our assessment and recommendations, in order of importance.

Update: *The Fiinu team has followed most of our recommendations and updated the contracts. The new version is at commit* `548e5ed6812da14623eeb30be64eee9bc45edb48`.

Critical Severity

Unbounded loops

There are several loops in the contract which can eventually grow so large as to make future operations of the contract cost too much gas to fit in a block. Additionally, using unbounded loops incurs in a lot of avoidable gas costs for all token transactions.

As an example, there is a state variable `allFNUHolders` which holds an array with the addresses of all token holders. When an address becomes a token holder, it is added to this array. Whenever an address is left with balance zero after a transfer, the array is iterated over to find the address and remove it (`removeAddress`). If the array is large enough, the iteration could cost too much gas and it would be impossible to make a `transfer` function call.

array. The same unbounded loop problem applies. This kind of imperative code-style is not a good fit for Ethereum smart contracts.

To fix this problem the two arrays have to be removed, and the profit sharing feature completely redesigned.

One suggestion for implementing these features securely is calculating the profits “on demand” for each token holder, instead of ahead of time for all of them. This would require the token to support taking a snapshot of all balances at the time profits are loaded into the smart contract. It might be worth considering using the MiniMe Token for this.

Update: *The token was migrated to use MiniMe Token, and the profit sharing functionality was reimplemented to use its features and avoid the problems mentioned here.*

Delayed price increase after target amount raised

The token price is meant to start increasing after the target amount of 100,000 ETH has been raised. However, due to the ordering of integer divisions the price doesn’t increase until actually 200,000 ETH have been raised. Line 207 reads `_wei.div(raisedWei.div(targetRaiseWei))`: notice that `raisedWei.div(targetRaiseWei)` will be 1 until `raisedWei` is `2*targetRaiseWei`, and the price adjustment won’t be made until then.

Replace the expression in line 207 for `_wei.mul(targetRaiseWei).div(raisedWei)` to achieve the desired price adjustment.

Update: *Fixed as suggested in* `91ff77c`.

High Severity

Staff allocations assigned to placeholder addresses

The staff allocations done in `Milestone_ICOSuccessful` mint tokens for the addresses `0x01`, `0x02`, `0x03` and `0x04`. These addresses are not owned by anyone and their tokens will not be usable. This might have been a temporary measure until the actual addresses were

Update: *Fixed in* `5295691`.

Profit distribution can be reset by owner at any time

There is a function `PrepareForProfitShare` callable by the owner at any time, which resets the distribution of profits: it brings every token holder's claimable profits down to zero. It's not clear what purpose it serves. Consider removing it, as it harms the trustlessness of the contract.

Fiat investments do not respect limits

Investments submitted by the admin via `investFIAT` do not respect the `maxRaiseWei` limit. Consider enforcing this by adding `require(raisedWei <= maxRaiseWei)` at the end of the function.

Update: *The function* `96e4c50`.

Medium Severity

State transition to "successful" doesn't check raised amount

The state transition from `ICOclosed` to `ICOSuccessful` doesn't check if the crowdsale succeeded or not. Consider adding the line `require(raisedWei >= minRaiseWei)` in `Milestone_ICOSuccessful`. Consider also adding similar requirements elsewhere, such as `require(raisedWei < minRaiseWei)` in `Milestone_ICOFailed`.

Update: *Fixed as suggested in* `55f9886`.

Low Severity

Lack of events for investments

For auditability purposes, especially given that there is an admin function with potential for abuse (`investFIAT`), it would be desirable to log events for investments. Consider emitting: 1) in the fallback function an event `Investment` with beneficiary address, amount of received ETH, and amount of tokens bought; 2) in `investFiat` an event `FiatInvestment` with the same data, plus the amount of fiat money invested or exchange rate used.

Update: *Fixed as suggested in* `040c66c`.

holdings: `balances[addr].mul(msg.value).div(totalSupply)`. It must be taken into consideration that integer division truncates the remainder. In this particular case, the amount lost to truncation will be negligible, and is in some sense necessary (you cannot transfer a fraction of a wei). However, a small improvement could be made by letting `balances[addr].mul(msg.value)` accumulate, and performing the division at the last moment, when the token holder claims his profit shares.

Keep in mind, too, that integer division is used to convert ether amounts to FNU amounts, and that due to truncation some amounts of ether will be received for no FNU compensation.

Possible erroneous Transfer event in transferFrom

In `transferFrom` it is not checked that the source account has enough balance. This is not severe because, in general, the safe subtraction will revert the transaction if more than the available balance wants to be transferred. However, in the special case that someone calls `transferFrom` with the source and destination as the same account, the same won't happen because the balance is first incremented, and only then decremented. The resulting balance will remain the same, but it would be possible for anyone to emit a `Transfer` event with themselves as destination and an arbitrary amount. This might be misinterpreted for a real transfer by an application or individual. Consider swapping the lines so that the balance is first decremented and then incremented, in order for such a transaction to fail.

Update: *Fixed as suggested in* `74e1172`.

ERC20 compliance

Although a minor problem, the `decimals` state variable should be defined with type `uint8` to be compliant with ERC20.

Additionally, there is a restriction to the usage of `approve` intended to mitigate a problematic race condition, but it is not ERC20 compliant. Consider removing the line, and including an alternative mitigation.

Update: *Fixed in* `767b11e`.

Notes & Additional Information

and `ICOFailed`. Consider making it consistent.

- The struct type named `Whitelist` might be more aptly named `WhitelistEntry`.
- In `manageAdmins`, it's not necessary to make a special case for `_add == false`, because `admins[_address] = false` is exactly equivalent to `delete admins[_address]` (including the gas reimbursement). Consider rewriting the function body simply to `admins[_address] = _add`. The same goes for `manageInvestors`.
- Solidity provides a shorthand notation for writing ETH amounts like `1 ether`. Consider using it to define the crowdsale limits (e.g. `minRaiseWei = 20000 ether`), and elsewhere like in the token fallback function.
- In the `Milestone_*` functions in the token contract (`Milestone_ICOSuccessful` and `Milestone_BankLicenseFailed`) it's not necessary to repeat the `inState` modifiers. They will be handled by the call to `super`.
- It's not a problem for the current token because the functions always return `true`, but when overriding `transfer` you should respect the return value of the call to `super`: if it returns `false`, the failure must be relayed. The same goes for `transferFrom`.
- `investFIAT` adds the investor address to the list of approved investors if it's not already in it.

Update: *Most suggestions were implemented in* `ba52c70`.

Conclusion

Two critical severity and three high severity issues were found and explained, along with recommendations on how to fix them. Some changes were proposed to follow best practices and reduce potential attack surface.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Fiinu Token contract. We have not reviewed the related Fiinu project. The above should not be construed as investment advice. For general information about smart contract security, check out our thoughts here.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits