OpenZeppelin

Fuel Token Audit

OPENZEPPELIN SECURITY | SEPTEMBER 22, 2017 Security Audits

The Vanbex team asked us to review and audit their Fuel Token (FUEL) and crowdfund contracts. We looked at the code and now publish our results.

The audited contract is in the etherparty/FUEL-Contracts repository. The version used for this report is the commit `3717b751bb2fa57ae300776a93ee4d7d7beb2c07`.

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

Critical Severity

Leftover presale supply can be left unrecoverable

When `FuelToken` is deployed, the total supply of 1 billion is divided into allocations destined for various purposes, including presale and crowdfund. The intention is for all of the supply to be eventually sold. Out of the presale and crowdfund allocations, tokens which aren't sold during the corresponding stage are meant to be eventually transferred to the platform address, which would later sell them via the Etherparty platform.

When the presale is finalized (via `finalizePresale`), its remaining allocation is transferred to the crowdfund. When the crowdfund is finalized (via `finalizeCrowdfund`), its remaining allocation is finally transferred to the platform. There is an *implied ordering* between these two calls which is not enforced in the smart contract code. As a consequence, the crowdfund could be mistakenly finalized before the presale, and any remaining tokens allocated for the presale would then be unrecoverable (for they would be transferred to the crowdfund address with no way to get them out anymore).

To prevent this, add the precondition `require(presaleFinalized)` to the `finalizeCrowdfund` function.

***Update:** Fixed in* `7a30774`.

High Severity

Error-prone code duplication

There is an extremely high level of code duplication in the contracts. Although it is not causing a problem in the current version of the code, it is highly likely that small revisions will create bugs.

For example, there are six different implementations of transfer (for example in `transferFromCrowdfund` and `releaseVanbexTeamTokens`), and there are other places where balances are modified with confusing semantics. In the case of transfers, consider using the contract's `transfer` function directly.

In general, be on the lookout for code duplication and consider refactoring the current code by creating internal helper functions with clearer semantics.

Update: *Regarding transfers, the team replied: "The* `transferFromCrowdfund` *and* `releaseVanbexTeamTokens` *methods have requirements not available in the* `transfer` *method. The crowdfund must be able to transfer before the tokens are able to transferred by the public, and the vanbex team tokens are not being decremented from a balance. We prefer this solution to cluttering up the transfer method itself." The* `FuelToken` *interface issue was fixed in* `91f8920`.

Presale balance may not be deliverable after crowdfund starts

The function `deliverPresaleFuelBalance` requires that the receiver have a balance of zero. There is a comment which states that all individual contributions will be aggregated, so that for each contributor the function will only be called once.

However, in the same way as the critical issue we reported, it is implied that the presale balance will be distributed before the crowdfund even begins. It could be the case that a user buys tokens both in the presale and in the crowdfund, and if the presale balance is not delivered before the latter, the presale delivery would fail. Furthermore, since presale deliveries will be made in batches, it would likely be very difficult to debug the cause of one batch failing.

Consider enforcing that all presale deliveries happen before the crowdfund starts, or lifting the requirement that an account should have no balance before transferring presale tokens to it.

Update: *Fixed in* `7a30774`.

Medium Severity

Unclear semantics for Transfer

The ERC20 token standard specifies the `Transfer` event to log transfers. The specification is informal, and there is not an exact characterization of when the event should be emitted. However, since this event is relied on by applications, it is in the interest of the project to try to respect the

The `Transfer` event is used throughout `FuelToken` in unexpected and inconsistent ways. When vested tokens are released (via `releaseVanbexTeamTokens`), a transfer from the token contract itself is logged. This is conflicting with the expected semantics of the event, because the token contract didn't have those tokens as part of its balance to begin with. In `transferFromCrowdfund`, the event is emitted with the crowdfund as the source, but there was never a `Transfer` event with the crowdfund as the destination.

Consider emitting this event with the zero address as the `from` field, which is the accepted way to log creation of tokens.

Update: *Fixed in* `759557a` *and* `a363180`.

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 in* `4a5ba4a`.

Unchecked return value

The function `finalizeCrowdfund` is defined to return a boolean value indicating success. However, this return value is ignored when the function is called. Since the return value is currently always `true`, consider removing it altogether. Otherwise, always check the return value.

Update: *The return value was removed in* `f85fc45`.

Duplicate data

value is error prone, as it could easily get out of sync. Consider getting the value directly from the crowdfund, as `crowdfund.endsAt()`.

Semantics of totalAllocatedTokens

It is unclear what the `totalAllocatedTokens` state variable of `FuelToken` means. At the moment the crowdfund is finalized, this variable will get out of sync with the amount of tokens actually distributed, because the platform will have all of the remaining supply to sell. Consider removing the variable altogether, or calling `allocateTokens` when the final remaining tokens are transferred to the platform in `finalizeCrowdfund`.

Update: *The variable was removed in* `86abce6`.

High coupling between token and crowdfund contracts

The token contract does a lot more than handle balances, transfers, and allowance. It has special code for crowdfund-originated “transfers”, for the Vanbex team vesting, and for the presale. It has variables and events to track crowdfund and presale state. As a result of all this, the code is very hard to follow and to ensure it is bug-free. At this stage, it would be a high risk to redesign the contracts, so we will not recommend that. However, the team should keep in mind to follow software engineering principles such as separation of concerns, single responsibility, low coupling, etcetera, for any new contracts they write.

Low Severity

Rejecting non-zero transfers is not ERC20 compliant

The ERC20 specification states that “transfers of 0 values MUST be treated as normal transfers”. In `FuelToken` these are rejected. Consider removing the modifier `nonZeroAmount(_amount)` from `transfer` and `transferFrom`.

Update: *Fixed in* `2b9e0b3`.

Notes & Additional Information

- Consider using OpenZeppelin’s `StandardToken` implementation to build on top of.

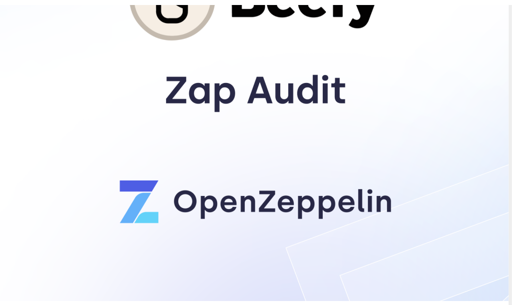
- The “price” of a token usually means how much ETH is necessary to buy one unit of it. In the contract it is used to mean the amount that can be bought with 1 ETH. This concept is usually referred to as “rate”.
  *(**Update:** Renamed to “rate” in `505439c`.)*
- This documentation comment in `NonZero` is incomplete.
- The fallback function in `FuelCrowdfund` has the `nonZeroValue` modifier added. This can be circumvented by a user by calling `buyTokens` directly. Consider placing the modifier in `buyTokens` instead. (The transaction would fail nonetheless because `transferFromCrowdfund` won’t accept an amount of zero tokens.) *(**Update:** Fixed in `9f1d6d6`.)*
- The function `closeCrowdfund` of `FuelCrowdfund` doesn’t itself require that it be called after the crowdfund period ends. It calls `finalizeCrowdfund` in `FuelToken`, however, which does fail and revert the whole transaction in that case. This is a strange assignment of responsibilities. It should be the crowdfund contract itself which rejects the transaction before the end of the period. *(**Update:** Fixed in `c4c6667`.)*
- The state variable `amountOfPublicTokensToAllocate` of `FuelToken` is merely the sum of `presaleSupply` and `icoSupply`, which is a constant value. It is not used for anything. Consider removing it as well. *(**Update:** removed in 4215e8c.)*
- It is undocumented and unclear what the `tokens` field of the `CrowdfundFinalized` and `PresaleFinalized` events should mean. In the implementation the value used is the amount left over from the crowdfund and presale allocations respectively. Consider choosing a better name, such as `remainingTokens`, and documenting its meaning. *(**Update:** Fixed in `df2bf81`.)*
- The empty fallback function is not needed because Solidity will by default reject value transfers. *(**Update:** Fixed in `d56da95`.)*
- Constant values can be declared as `constant` state variables and initialized in the declaration itself (as opposed to in the constructor). Consider doing this for `vanbexTeamSupply`, `platformSupply`, `presaleSupply`, etcetera.
- Consider writing token amounts parameterized by the `decimals` state variable. For this, it can be helpful to define a variable `uint256 constant FUEL_UNIT = 10 ** uint256(decimals)`. Afterwards, you can write, for example, `vanbexTeamSupply = 50e6 * FUEL_UNIT`. This is clearer and more maintainable.

Conclusion

One critical severity and two 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 Fuel Token contract. We have not reviewed the related Etherparty 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

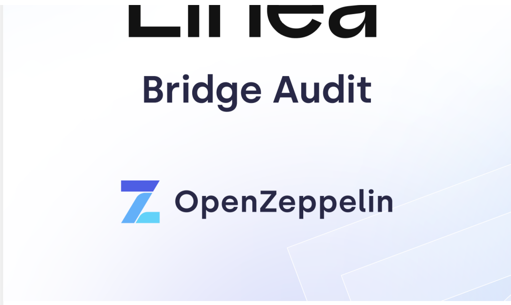
BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

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

OpenZeppelin

Defender Platform

Secure Code & Audit
Secure Deploy
Threat Monitoring
Incident Response
Operation and Automation

Services

Smart Contract Security Audit
Incident Response
Zero Knowledge Proof Practice

Learn

Docs
Ethernaut CTF
Blog

Company

About us
Jobs
Blog

Contracts Library

Docs