Tierion Presale Audit

OPENZEPPELIN SECURITY | JULY 25, 2017

Security Audits

The Tierion team asked us to review and audit their Tierion Presale code. We looked at the contract and now publish our results.

The audited contract is in the file TierionPresale.sol with SHA-1 hash `e9462ae354e2b4acbd75ff46c7dc1c476a1d1925`. The version of OpenZeppelin used is 1.1.0.

The code is very lean and well commented. Good work reusing code!

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

UPDATE: *Tierion has implemented all of our suggestions and fixed all issues. We have found no problems in the updated code. The new TierionPresale.sol file has SHA-1 hash `c7cb106b0627bfb3f1f35b019968451947fa0081`. We have added a notice to each entry accordingly.*

Severe

Unsafe use of Destructible

As seen in line 7, the contract is `Destructible`: the owner can destroy it at any moment, forwarding the funds to an address. It should be first noted that similar functionality is provided by the contract's own `withdraw` function, insofar as all of the contract's funds can be forwarded to the owner. There are two problems with this.

Withdrawing a contract's balance, however, is not `destroy`'s *raison d'être*. It is removing the smart contract's code from its address. A huge implication of this is that after a contract is destroyed, transactions sent to the address will no longer be interpreted as function calls, but as simple ETH transfers. **Ether sent to a destroyed contract is effectively lost forever.** We strongly recommend against making any contract whose purpose is to receive payments an instance of `Destructible`. Someone could send a payment after destruction without knowing their payment won't be received and their money would be lost.

Consider not inheriting from `Destructible`, and using the contract's `withdraw` function for withdrawal.

UPDATE: *Our recommendation was followed and the contract is no longer* `Destructible`*. In this way the contract is no longer vulnerable to the potential loss of Ether described above.*

Potential Problems

Receipt event has confusing semantics

The `Receipt` event is emitted, in line 29, for each payment processed. It logs the investor address, payment hash, and amount of ether received. The same event is emitted, with an empty hash, when the owner withdraws the collected ether in line 37. This overloading of the event's semantics seems confusing. Consider emitting instead a separate `Withdrawal` event in `withdraw`.

UPDATE: *The event has been removed from the* `withdraw` *function. It is now only emitted for payments, which are clearer semantics.*

Warnings

Avoid using var to define variables

There are uses of `var` in the code, where the type of the variable is inferred from the expression on the right hand of the definition. We recommend avoiding this feature because in some cases it might infer a smaller integer type than the developer might think. It is best to be explicit regarding types.

It is customary for a mapping's name to describe the *value_s associated to its keys. In the case of the* `hash` *state variable defined in line 15, the name describes the mapping's _keys* instead. Since the purpose is to map hashes to the amount of ether received associated to each hash, consider renaming the state variable to `weiReceived`, for example.

UPDATE: *Following our recommendation, the variable has been renamed* `weiReceived`.

Use safe math

There is only one math operation in the contract, in line 28. It's always better to be safe and perform checks for overflow. Consider using OpenZeppelin's safe math library.

UPDATE: *Following our recommendation, the math operation mentioned above has been secured with an overflow check.*

Solidity version

The contract requires version 0.4.11 of Solidity. It should be noted that 0.4.13 was released a few days ago. Consider changing the solidity version pragma to the latest version (`pragma solidity ^0.4.13;`) to enforce latest compiler version to be used.

UPDATE: *For external reasons concerning the tooling used, the Solidity version was kept at 0.4.11. Despite not following the recommendation there is no security risk incurred.*

OpenZeppelin version

The project uses version 1.1.0 of OpenZeppelin. It should be noted that 1.2.0 was released a few days ago. Consider changing the OpenZeppelin version to the latest in the project's package.json.

UPDATE: *Following our recommendation, OpenZeppelin was updated to version 1.2.0.*

Notes and Additional Information

- Good job using OpenZeppelin!

Conclusions

problems in the upgraded code.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Tierion Presale contract. We have not reviewed the related Tierion project. The above should not be construed as investment advice or an offering of tokens. 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

© Zeppelin Group Limited 2023