WAX Token Audit

OPENZEPPELIN SECURITY | DECEMBER 19, 2017

Security Audits

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

The audited code is located in the waxio/wax-erc20-delivery-contract repository. The version used for this report is commit `6c7098dd2522630d74c9600f678b3b28d58fa9aa`.

Here is our assessment and recommendations, in order of importance.

Update: *The WAX team has followed our recommendations and updated the WAX Token contract. The new version is at commit* `b545b4adb093daa23c2bb013b2ac8fe91753739a`.

Critical severity

No critical severity issues were found.

High severity

No high severity issues were found.

Medium severity

No medium severity issues were found.

Low severity

Install OpenZeppelin via NPM

have been copied from the [illegible] branch, instead of the release. This violates OpenZeppelin's MIT license, which requires the license and copyright notice to be included if its code is used, and makes it difficult and error-prone to update to a more recent version.

Consider following the recommended way to use OpenZeppelin contracts, which is via the zeppelin-solidity NPM package. This allows for any bugfixes to be easily integrated into the codebase.

Update: *Impoted OpenZeppelin as NPM dependency in* `57d8603`.

OpenZeppelin standard contracts were modified

Additionally to copying OpenZeppelin's contracts instead of installing them via NPM, some of them were modified.

The contract `BasicToken` was modified to add a state variable `contractAddress`, and the functions `transfer`, and `transferFrom` in `StandardToken` were modified to require that the receiver of tokens is not the `WaxToken` contract itself.

This is not the way OpenZeppelin standard contracts should be used. Making changes to open-source libraries, instead of using them as is, can be dangerous and won't allow you to integrate bug-fixes into the codebase easily.

Add these additional preconditions to the `WaxToken` contract, instead of OpenZeppelin's contracts. For example, override the `transfer` function adding the additional restriction and call `super.transfer(…)` within it. (Note: this has actually already been done, so it's a matter of removing the modifications to OpenZeppelin's contracts.)

Update: *Fixed in* `57d8603`.

Token allowances may be increased or decreased when paused

The `StandardToken` contract defines an `increaseApproval` and `decreaseApproval` methods that could be called and executed even when the `WaxToken` contract is paused.

Notice that this is already implemented in OpenZeppelin as `PausableToken`. Consider using it instead.

Update: *Fixed as suggested in* `fb1d356`.

Notes & Additional Information

- The added fallback function with a manual `revert` is not necessary, since this is the default behavior of Solidity contracts. Consider removing it.
- The `WaxToken` contract is requiring that the receiver is not the zero address in `transfer` and `transferFrom`. OpenZeppelin's `BasicToken` and `StandardToken` already have this precondition. Consider removing those validations from the `WaxToken` contract.
- In the `WaxToken` constructor, a reference to `this` (the address of the contract) is stored in a variable called `contractAddress`. We see no reason why this variable is used instead of directly using the value of `this` throughout the contract.
- The functions `transfer` and `transferFrom` have a precondition to reject a transfer of tokens to the `WaxToken` contract itself. It would be good to define a modifier that ensures this condition to avoid code duplication.
- The `WaxToken` contract defines all its public functions without the `public` keyword explicitly. It's important to mark `public` functions as such, even though Solidity functions are public by default, to avoid confusion. Moreover, since version 0.4.17 of Solidity, the compiler will show a warning if the visibility specifier (`public`) is not explicitly given.
- The `INITIAL_SUPPLY` variable is initialized multiplying `1000000000` by the corresponding amount of decimals defined for the contract. Such a long number is hard to verify manually. It would be better to use the notation `1e9` to represent this kind of number in order to avoid typos.
- Keep in mind that there is a possible attack vector on the `approve` / `transferFrom` functionality of ERC20 tokens, described here. Consider using the mitigations implemented in OpenZeppelin's `StandardToken`.

Update: *Suggestions were implemented in* `841b4f9`, `72686ef`, `d597238` *and* `2d9ffbc`.

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 WAX Token contracts. We have not reviewed the related WAX 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