ArroToken Audit

OPENZEPPELIN SECURITY | JANUARY 13, 2020

Security Audits

The Arro team asked us to audit their ERC20 ArroToken project. We look at the code and now we publish our results.

The audited code is located in the pjsimpkins/ArroERC20Code Github repository. The version used for this report is commit `1ca1cc85f2e7e046070c187031b75b3335ef57c3`.

with first round of fixes]". Our analysis of the mitigations assumes the pull request will be merged, but disregards all other unrelated changes to the code base.

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

Critical severity

No critical severity issues were found.

High Severity

[introduced with the first round of fixes] Stray tokens can be locked in ArroToken contract.

The `transferAnyERC20Token` function is intended to transfer any ERC20 tokens accidentally received by the `ArroToken` contract. However, this function calls the `_transfer` internal function passing the address of the stray ERC20 token as the sender. This means that instead of transferring stray tokens out of the `ArroToken` contract, this function tries to transfer the `ArroToken`s owned by the stray contract.

Consider calling the transfer function of the stray token contract, instead of the one from the `ArroToken` contract.

Not following good Smart Contract development practices

The ArroToken project lacks of a standard Solidity project structure. There are no unit tests and integration tests implemented, and there is no code coverage set up. All the contracts are defined in the same file, which doesn't have the proper file extension. Additionally, the `README.md` file is empty.

Consider using the OpenZeppelin SDK or Truffle for setting up the project environment, organizing the contracts, and adequately develop the project.
Consider using the OpenZeppelin Test Environment and OpenZepplin Test Helpers for writing automated Smart Contract tests.

Update: *Partially fixed. Arro's statement for this issue:*

> The project has been restructured to follow the standard Solidity project structure provided by the OpenZeppelin CLI. The code functionality relies heavily on the provided libraries and interfaces from the OpenZeppelin SDK, this provides test coverage over a majority of the functionality.
> There are plans in the future to add unit or integration tests on contract functions.
> A relevant README has been added that provides general project information and installation and development instructions.

Note that the project is missing the `.openzeppelin/project.json` *file from the OpenZeppelin CLI.*

Medium severity

No allowance front-running mitigation

The `ArroToken` contract is vulnerable to the ERC20 approve and double spend front-running attack.
In this attack, a token owner authorizes another account to transfer a specific amount of tokens on their behalf, and in the case that the token owner decides to change that allowance amount, the spender could spend both allowances by front running the allowance-changing transaction.

Consider using the OpenZeppelin's ERC20 implementation, and the decreaseAllowance and increaseAllowance functions to help mitigate this.

Update: *Fixed in pull request #1.*

Outdated solidity version in use

An outdated Solidity version, `0.4.24`, is currently in use.

This can introduce vulnerabilities to the project that were fixed in newer solidity versions, such as the short address attack in the `transfer` and `transferFrom` functions.
In this attack, a user inputs a malformed address as destination for a token transfer in an exchange, which is misled to craft a transaction for a much bigger amount of tokens. This does not

when interacting with the token contract.

Consider bumping the project to the latest version supported by OpenZeppelin Contracts (presently 0.5.15).

Update: *Fixed in pull request #1.*

Missing docstrings

All the contracts and functions in Arro's code base lack proper documentation. This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, docstrings improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Consider thoroughly documenting all functions (and their parameters) that are part of the contracts' public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Partially fixed in pull request #1. Arro's statement for this issue:*

> The project structure has been changed to rely heavily on the OpenZeppelin ERC20 implementation. This provides a well-documented base of most of the contract functionality. The rest of the structure of the documentation has been changed to follow a consistent structure but will be updated in the future to follow the Ethereum Natural Specification Format (NatSpec).

Low severity

Lack of event emission `OwnershipTransferred` on contract construction

The `OwnershipTransferred` event is not being emitted when the `ArroToken` contract is initialized.

This means that there will be no record of who was the initial owner of the token in the `OwnershipTransferred` event logs.

Update: *Fixed in pull request #1.*

Lack of event emission after modifying `newOwner` variable

In line 90 and in line 96, the storage variable `newOwner` is being modifed, but no events are being triggered.

This means that there will be no record of the changes of the `newOwner` variable, and traceability of the potential new owners of the `TokenArro` will be lost.

Consider to define an event for tracking the `newOwner` variable value, and emit an event each time it is modified.

Update: *Fixed in pull request #1.*

`owner` and `newOwner` storage variables can be modified without events emission

The `owner` and `newOwner` storage variables in the `Owned` contract are defined as `public`.
Even though they are not being modified within the `ArroToken` contract, there are no explicit validations for avoiding to assign them directly, which will bypass the emission of the `OwnershipTransferred` event.

Consider using OpenZeppelin's Ownable contract which already covers this, or consider declaring `owner` and `newOwner` variables as private, and add getter functions for accessing them.

Update: *Fixed in pull request #1.*

Duplicate code in `transfer` and `transferFrom` functions

The `transfer` and `transferFrom` functions share very similar code.
Duplicate code is more difficult to maintain, as it is longer and needs to be updated in different sections of the codebase.

Update: *Fixed in pull request #1.*

Wrong usage of inheritance with `Safemath` contract

A SafeMath contract is implemented in order to manage arithmetic overflows and underflows.

In this contract, all its functions are defined as public, which means that the `ArroToken` contract will expose them as part of its API, as it inherits from `SafeMath`. This extends the functionality that should be exposed by an ERC20 token. Additionally, the gas costs of the deploy will be higher as opposed to using `SafeMath` as a library.

Given that the functionality of the `SafeMath` contract can be exposed as a library, consider using OpenZeppelin's SafeMath

Update: *Fixed in pull request #1.*

The `TokenArro` contract can be transferred to the zero address

The `transferOwnership` function in the `Owned` contract does not prevent to transfer the ownership of the contract to the zero address.

Consider using OpenZeppelin's Ownable contract which already covers this, or consider restricting the new owner to non-zero addresses.

Update: *Fixed in pull request #1.*

Tokens can be transferred to the zero address

The `transfer` and the `transferFrom` functions in the `ArroToken` contract do not prevent to transfer tokens to the zero address.

Consider using the OpenZeppelin's ERC20 implementation which already covers this, or consider restricting the `to` parameter to non-zero addresses.

Update: *Fixed in pull request #1.*

The `totalSupply` function does not return the real total supply

This could lead to a misinterpretation of the real total supply of the token when querying it.

Consider modifying the the `totalSupply` function implementation so it returns the real total supply value, and consider using the OpenZeppelin's ERC20 implementation that implements the `_burn` and `_burnFrom` functions to keep the total supply variable up-to-date.

Update: *Fixed in pull request #1.*

Notes & Additional information

[introduced with the first round of fixes] Unnecessary inheritance and imports.

The `ArroToken` contract inherits from the `OwnableUpgradeSafe`, `Initializable`, and `ContextUpgradeSafe` contracts. Because the `OwnableUpgradeSafe` contract inherits from the `Initializable` and `ContextUpgradeSafe` contracts, the declaration to inherit from these two contracts is redundant.

Consider simplifying the code by removing the redundant inheritance declarations.

[introduced with the first round of fixes] `initialize` function missing the `initializer` modifier

The `initialize` function of the `ArroToken` contract is not using the `initializer` modifier. Currently this is not an issue because the functions called by `initialize` have the `initializer` modifier and will prevent it to be called more than once. However, it is safer to use the `initializer` modifier to make the intention clearer and safer in case the code is modified.

Consider adding the `initializer` modifier in the definition of the `initialize` function.

[introduced with the first round of fixes] Unnecessary calls to initializers

The `initialize` function of the `ArroToken` contract calls `__Context_init_unchained()`, `__Pausable_init_unchained()`, and

Consider calling `__ERC20Pausable_init()` to simplify the code.

Not following a consistent coding style

The code base deviates from the Solidity Style Guide. A consistent coding style helps with the readability of the project. Consider enforcing a standard coding style with help of linter tools such as Solhint.

Update: *Fixed in pull request #1.*

Multiple variables declared as uint

There are several variable declarations of `uint` variables throughout the project.

To favor explicitness, all instances of `uint` should be declared as `uint256`. See for example lines 25 and 29, 49, 109 and 111.

Update: *Partially fixed in pull request #1. In line 46 of `ArroToken.sol` there is still a `uint` declaration.*

No explicit visibility definition of variables

In line 111 and line 112, the `balances` and `allowed` variables are not explicitly declared as public.

To favor explicitness and readability, consider adding the visibility in all variable declarations.

Update: *Fixed in pull request #1.*

Link reference to the ERC20 EIP is deprecated

The links to the ERC20 EIP in line 46 and in line 161 references to an old ERC20 document.

Consider referencing the official ERC20 EIP site instead.

Update: *Fixed in pull request #1.*

Unused variables defined in function definitions

variables, and adding the necessary return statements when appropriate.

Update: *Fixed in pull request #1.*

Missing error messages in `require` and `revert` statements

There are `require` and `revert` statements without an error message specified in line 85, line 93, and line 216.

Consider adding an appropriate message describing the validation or revert reason.

Update: *Fixed in pull request #1.*

Inconsistent variable naming

Variable naming is inconsistent. For example, in line 109 `_totalSupply` has a leading underscore, while the other storage variables within the same contract do not.

Consider defining a naming convention based on Solidity Style Guide and follow this convention throughout the project.

Update: *Fixed in pull request #1.*

Non-standard initialization of ERC20

In the `ArroToken` constructor, the variables of the token are being hardcoded.

Consider using OpenZeppelin's ERC20Detailed contract's constructor for initializing the `name`, `symbol`, and `decimals`, and consider sending `totalSupply` as a parameter to the `ArroToken` constructor.

Update: *Fixed in pull request #1.*

Conclusion

No critical and one high severity issue was found. Some changes were proposed to follow best programming practices and reduce the potential attack surface.

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