Arcade City (ARC) token audit

OPENZEPPELIN SECURITY | OCTOBER 29, 2016

Security Audits

We've been asked by the Arcade City team to review and audit their new token code. We conducted the research and now publish our results.

The audited contracts are at their ac-token GitHub repo. The version used for this report is commit dd4207e1538f96eb3cbbf9e714eb38015bfe7c5a. The main contract file is ARCToken.sol.

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

Severe

We have not found any severe security problems with the code.

Potential problems

General code quality

Code quality of the ac-token project is low, which made auditing it hard. We recommend a refactor to improve code quality (more recommendations given next), and maybe doing a second security audit. Simpler code makes functionality more apparent and reduces attack surface. Given the high degree of test coverage, making changes to improve code quality will have a very low risk of introducing regressions.

Use latest version of Solidity

have `msg.value > 0`, which can cause the balance of the contract to be non-zero (which doesn't seem to be desired, based on the `multisig` forwarding account).

We recommend:

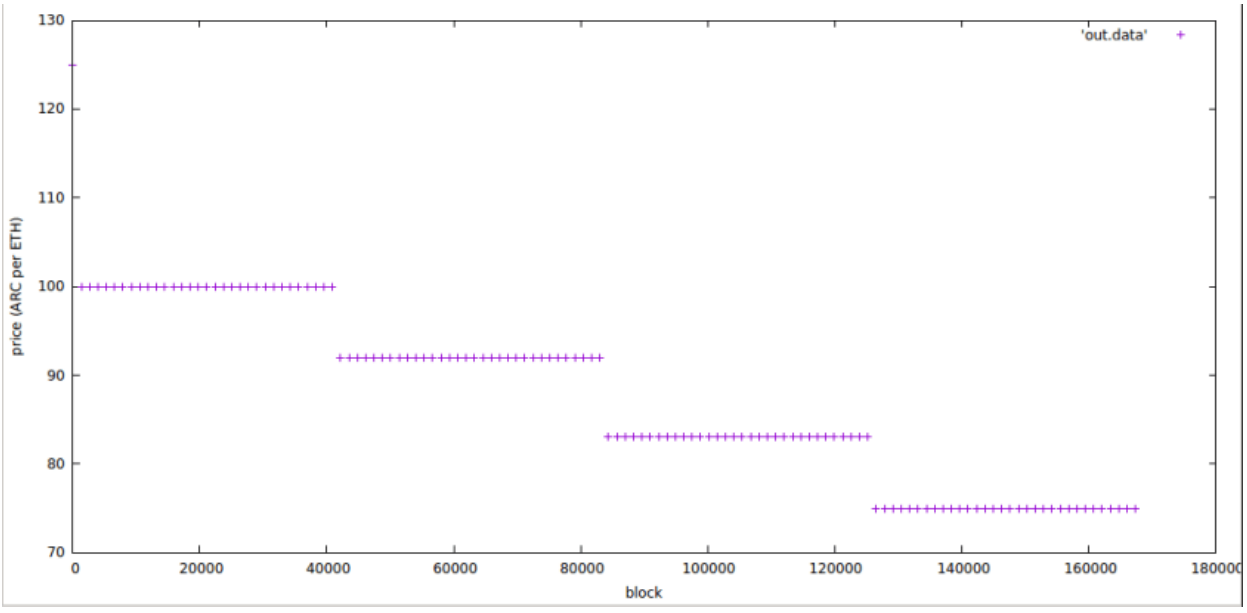
- Adding `pragma solidity ^0.4.2;` to the top of each contract file. (edit: fixed by ArcadeCity team)
- Adding `payable` function modifier to the fallback function andthe `buyRecipient` function. These are the only functions that need to be able to handle incoming payments.

Remove unneeded `call.value()`

Using `call.value()` is potentially dangerous, and was responsible for the TheDAO hack. We couldn't find a reason to use `multisig.call.value(msg.value)` instead of the simpler `multisig.send(msg.value)`. We recommend making this change. (edit: fixed by ArcadeCity team)

Price function is not linear

The code seems to intend a linear price function, but this is the actual behavior:



There is an initial period where 1 ether = 125 ARC, and then four steps of decreasing price. We recommend reviewing if the price curve meets the desired shape.

There are several magic constants in the contract code. Some examples are:

- https://github.com/ArcadeCity/ac-token/blob/dd4207e1538f96eb3cbbf9e714eb38015bfe7c5a/contracts/ARCToken.sol#L188-L199
- https://github.com/ArcadeCity/ac-token/blob/dd4207e1538f96eb3cbbf9e714eb38015bfe7c5a/contracts/ARCToken.sol#L267-L276

Use of magic constants reduces code readability and makes it harder to understand code intention. We recommend extracting magic constants into contract constants.

Remove unnecessary code

The `uint public presaleEtherRaised` variable seems to be unnecessary. Consider using **multisig.balance** in its place unless funds in the multisig address will be moved (and thus balance changed) before `endBlock`. Having unneeded extra variables and code increases risk and attack surface for contract's invariants to be broken.

Remove duplicate code

Duplicate code makes it harder to understand the code's intention and thus, auditing the code correctly. It also increases the risk of introducing hidden bugs when modifying one of the copies of some code and not the others. We recommend the following to remove duplicate code:

- Extract **StandardToken**, **SafeMath** and **Token** code into separate files, and import them from main contract files, instead of having exact copies of each in every file. We recommend using OpenZeppelin's implementation of StandardToken.
- `price` and `testPrice` functions repeat the same code. `price` could do `return testPrice(block.number);`. (edit: fixed by ArcadeCity team)
- `msg.sender` checks for authentication repeat the same patterns in many functions. Examples are this, this and this. We recommend extracting those to function modifiers.
- `buy` function seems redundant to the fallback function for **ARCToken.sol**. (edit: fixed by ArcadeCity team)

Remove commented code

Commented code just adds clutter for the reader and creates unncessary confusion. Remove the commented `allocateBountyAndEcosystemTokens` function. (edit: fixed by ArcadeCity team)

Tests should be independent

Some tests depend on each other. This makes testing specific functions in isolation more difficult. We recommend making each test independent by using a new token contract for each test case, instead of using the same token instance for every test.

Avoid name reuse

The **sendTokens** function in TokenVesting.sol reuses the name `vestAmount`, which is also a contract public variable. Using the same name for two different things is confusing and can bring unintended behaviors.

Additional Information and notes

- This comment is wrong. It should say “it should only continue if endBlock has passed OR presaleEtherRaised *has* reached the cap”
- `owner` can change the `multisig` address, and thus, has full control of where the contract funds go, even after crowdsale starts. (edit: fixed by ArcadeCity team)
- `price` function should not work outside of the [startBlock, endBlock] period. Line 190 in ARCToken.sol is not used, given that the `price` function is only called from `buyRecipient`, and `buyRecipient` only works in that interval. We recommend removing that line, and adding precondition checks to `price`.
- Using block heights is preferred to using timestamps for time-related logic. OK
- Some code indentation is inconsistent. For example, see this and this.

Conclusions

No severe security issues were found. Some changes were recommended to follow best practices and reduce potential attack surface. Above all, we recommend a refactor to improve code quality.

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

Threat Monitoring
Incident Response
Operation and Automation

Zero Knowledge Proof Practice

Blog

Company

About us
Jobs
Blog

Contracts Library

Docs