ABDK

TIMELOCK
Smart Contract

Mikhail Vladimirov and Dmitry Khovratovich

24th August 2018

This document describes the audit process of the Timelock smart contract performed by ABDK Consulting.

1. Introduction

We've been asked to review the Timelock smart contract given as a file.

Our major concernt is that contract's hierarchy seems to be incorrect. It does not allow implementing ether timelock without duplicating most of the code.
Correct hierarchy should be:

```
contract AbstractTimeLock {
// Implements basic logic,
// Contains abstract function that sends locked assets to
user
}

contract EtherTimeLock is AbstractTimeLock {
// Overrides abstract function with code that sends ether
// Add function that accept ether
}

contract TokenTimeLock is AbstractTimeLock {
// Overrides abstract function with code that sends
tokens
// Add function that accepts tokens
}
```

Other issues are listed below

2. TimelockERC20Onepager

In this section we describe issues related to the smart contract defined in the TimelockERC20Onepager.sol.

2.1 Major Flaws

This section lists major flaws which were found in the smart contract.

Line 167: the code `_value = msg.value` is unreachable.

2.2 Moderate Flaws

This section lists major flaws, which was found in the smart contract.

Line 169: looks like ether sent along with transaction into this payable method will stuck forever. It would be better to `add require (msg.value == 0);` here.

2.3 Documentation and Readability Issues

This section lists documentation issues, which were found in the smart contract.

1. Line 50,74: prefix `I` suggests this is an interface, but contract itself does not look so.
2. Line 51: using enum in this lines would make code more readable and less error-prone.
3. Line 92: the field `utilityToken` should be initialized in constructor of this contract, rather then derived contract. This will make code more readable and less error-prone.
4. Line 159: the mapping `mapping (address => mapping(uint => uint))` has two keys which meaning is unclear with documentation comment.

2.4 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 4: probably it would be better to allow owner to withdraw unaccounted tokens and ether from contract's balance.
2. Line 55: the internal method `getCassetteSize` is never called.
3. Line 56: types `uint256` and `uint` are intermixed in the code, while they mean exactly the same.
4. Line 58: pure method with no parameters is basically a constant. Why to define it as a method?
5. Line 92: the variable `address` should be of type IERC20.

6. Line 142: the event `OwnershipTransferred` will be emitted even if new owner is the same as current owner.
7. Line 154,155: indexing events by timestamp looks useless, because indexed fields allow querying by exact values only, but not by ranges.
8. Line 182: the value `_timestamp.length` already inside local variable `_len`. It could be used in this line.
9. Line 184,185,186: variables `_curValue`,`_curTimestamp` and `_subValue` used only inside the loop, so they can be defined there.
10. Line 202,206: perhaps, functions `release` and `releaseForce` should be public.

2.5 Unclear Behaviour

This section lists issues of the smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision.

1. Line 56,57,58,75: internal methods in this lines only called from derived contract. Why to define it in this line?
2. Line 174,194: operations `add(_value)` and `_curValue.sub(_subValue)` suggests that `transferFrom` delivers exactly as many tokens as was asked, which is not the case for fee-charging tokens. Some comments should be added regarding those or additional controls should be added.
3. Line 202: the function `release` does not process `CT_ETHER` case. Is it OK?

2.6 Arithmetic Overflow Issues

This section lists issues of token smart contract related to the arithmetic overflows.

Line 18,45:the function `assert(c / a == b)` relies on undocumented overflow behavior in order to detect overflow after it happened. It would be better to prevent overflow by checking arguments before multiplying and adding them.

2.7 Other Issues

This section lists stylistic and other minor issues which were found in the token smart contract.

1. Line 55,56,57,58,75,154,155,180: Solidity code style does not imply underscore in the function name.
2. Line 56, 57: what does mean `abstract` in `acceptAbstractToken` and `releaseAbstractToken`?
3. Line 63, 64, 65, 67, 68,154,155: method parameters in this lines are without underscore.

4. Line 63, 64, 65, 67, 68: defining methods in this lines as `external` looks odd. `View` methods usually do not try to read `msg.sender` or `msg.value` or other transaction metadata that is affected by calling convention used to invoke method. Should probably be `public` as in ERC20 standard itself.

3. Our Recommendations

Based on our findings, we recommend the following:

1. Fix major and moderate issues which could damage the business logic. Also fix the hierarchy of the contracts.
2. Simplify the code, improving its readability.
3. Fix arithmetic overflow issues.
4. Check issues marked “unclear behavior” against functional requirements.
5. Refactor the code to remove suboptimal parts.
6. Fix the documentation and other (minor) issues.