ABDK

World Wide Health
Smart Contract: Review

Mikhail Vladimirov and Dmitry Khovratovich

1th December, 2017

This document describes the issues, which were found in World Wide Health smart contract during the code review performed by ABDK Consulting.

1. Introduction

We were asked to review a contract ZMINE.

2. World Wide Health

In this section we describe issues related to the smart contract defined in the World Wide Health.sol.

2.1 Critical Issues

The contract under investigation has two critical issues, which make it vulnerable to a wide group of attackers.

1. Line 362 and 368, 402, 428: method `setAirDropDestination` and `holdersList` (in three last lines) gives a permission to anyone to make the `holdersList` array gigantic. As a result, method `holdersList` and, more importantly, `airdrop` will not work if holdersList is too big (Line 437, 455, 458 and 472).
2. In line 629: function `transferFor` is public. It means that the function can be called by anyone, and he can then steal all presale tokens. Probably `onlyOwner` is missing here.

2.2 EIP-20 Compliance Issues

This section lists issues of token smart contract related to EIP-20 requirements.

1. Line 158, 169: parameter names are different from those that are defined in EIP-20. This maight cause some compatibility problems.
2. Line 391, 418: condition `_value > 0` directly violates EIP-20.
3. Line 498: instead of `uint` there should be `uint8` (according to EIP-20).

2.3 Documentation and Readability Issues

This section lists documentation issues, which were found in the token smart contract, and the cases where the code is correct, but too involved and/or complicated to verify or analyze.

1. Line 326: it is mentioned in the comment that `index if holder in holdersList`. But if index is guaranteed to be not equal to zero, `holders` mapping is not unnecessary.
2. Line 329: comment is confusing. Who should set to zero and what exactly should he set?
3. Line 331: the variable name and comment look completely unrelated to each other.
4. Line 333: the variable name `_address` is confusing. Probably there should be `sender`/`author`/etc.
5. Line 420-423: code mentioned here is already implemented in `StandardToken`, so it would be better to call `StandardToken.transferFrom` in this case.
6. Line 568, 586,630, 640,658: probably instead of `>` there should be `>=`.
7. Line 569, 590: the bracket pair is redundant here.
8. Line 77: event `AuthorizationSet` should be renamed to `AuthorizationChanged` to improve the readability.
9. Line 351: instead of `holders[_address] == false` — `!holders[_address]` would be more readable.
10. Line 543: a regular date representation of number `1515376800` in a comment would improve the readability.
11. Line 612, 613: need human-readable comment with date.

2.4 Arithmetic Overflow Issues

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

1. Line 13, 31: the correctness of this check relies on how an overflow works in Solidity, but this behavior is not documented, so it may change in the future. We recommend to redevelop the function so the overflow does not ever happen.
2. Line 394: the comment is incorrect. Usually, `SafeMath` is used as the second line of defense. Related on that fact, the business logic of the smart contact is designed in such a way that over(under)flow is never possible, so `SafeMath` just checks that over(under) flow never happens. That's why SafeMath uses `assert`, not `require`. Making `SafeMath` to be a part of the business logic itself leaves the contract with only a single line of defense.
3. Line 512: in the subtraction underflow is possible.
4. Line 562: here might be a possible overflow that allows to actually decrease hard cap.
5. Line 563: overflow is possible.

2.5 Unclear Behavior

This section lists issues of the token 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 187, 227, 390, 417: the condition `_to != address(0)` is not necessary. Addresses like 0x0 can be used to emulate token burn.
2. Line 342: function `setTreasureBox` is available for everyone. The consequences of this avialability are not clear.
3. Line 376: method `setAirDropDestinationAndApprove` does not push `_destination` into holders list, unlike `setAirDropDestination`. Perhaps, it should be replaced.
4. Line 448, 532: perhaps `address` should be indexed.
5. Line 572, 644: `1 ether` seems to represent `10^18` rather than the unit of currency. This representation lowers the readability and is error-prone. Also, div here may cause rounding errors which might accumulate over time.
6. Line 631: perhaps, `require(_amount >= minTx && _amount <= maxTx)` restriction is not needed.
7. Line 702: using `transferFrom` assumes that the owner has approved the `FounderThreader` contract with tokens. Is this an adequate behavior?

2.6 Suboptimal Code

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

1. Line 11: putting the rest of the method into `false` clause will the fact that the rest of the method is conditionally executed more obvious.
2. Line 63: if `newOwner` is zero, then the method `if (newOwner != address(0))` executes successfully, but this executions silently does nothing in particular. In such case it would be better to revert, so the caller knows that owner hasn’t been changed.
3. Line 77, 115: it would be more efficient and more convenient to have two separate events: first one to be logged when the address becomes authorized, and the second one for the case when the address becomes unauthorized.
4. Line 84: `AuthorizationSet` event is not logged here, which makes it harder to track all authorized addresses for a contract.
5. Line 100:it is a common practice to do nothing in case when the new set value is the same as the current one being used. Situation does not look bad enough to revert.
6. Line 115: probably, `addressWhiteListed` parameter should be indexed.
7. Line 122: `whiteListSet` event is not logged here, which makes it harder to track all whitelisted addresses.
8. Line 133: `isWhiteListed` method would not be necessary if `whiteListed` mapping is declared as `public`.
9. Line 142: it is a common practice to do nothing in case when new set value is the same as current one being used. Situatuion does not look bad enough to revert.

10. Line 217: a default access modifier `internal` for storage variables, so this declaration does nothing effective. Also, other internal storage variables in the same contract do not have it.
11. Line 311 and 575, 634, 647:a return value is not checked in `token.transfer` and `transferFrom`.Probably it would be better to check and revert in case transfer fails.
12. Line 334: it would be more efficient and more convenient to have two separate events: one being logged when the address is made to be and is an exchanger, and another one when the address becomes not an exchanger (`_isExchanger`).
13. Line 338-339: the construction here can be reduced to `holderList [holderList.length++] = owner;`.
14. Line 347: it as a common practice to do nothing in case when the new value is the same as current value bieng used. Situation is does not look bad enough to revert.
15. Line 352: `if` operation is being performed at multiple places.Probably, you should consider moving it into a separate method.
16. Line 354-356: the construction here can be reduced to:
    ```
    holderIndex [_address] = holderList.length;
    holderList [holderList.length++] = _address;
    ```
17. Line 377,378: probably, it would be better to just call `approve` and `setAirDropDestination` respectively to reduce the code duplication.
18. Line 395-397: the code listed here is already implemented in StandardToken. It would be better to just call `StandardToken.transfer` here.
19. Line 436: the usage of `getHolders is incorrect.` Usually arrays are not returned as a whole, so two separate methods are implemented instead: one to get an array length and another one: to get i-th element of an array. This is what Solidity would generate if `holdersList` is declared as public.
20. Line 450: gas consumption of `airDrop` linearly depends on the number of token holders. It is easy to increase the number of token holder for making this function not to fit into block gas limit. A different approach should be chosen.
21. Line 461: `1000 ether` in condition is confusing. TokenHolding is not in Wei.
22. Line 477-480: code here is already implemented in StandardToken, would be better to just call `StandardToken.transfer` here.
23. Line 500: `1000000000 ether` is confusing. TotalSupply is not in Wei.
24. Line 520: there is no need to revert `require(rate != _rate)` here.
25. Line 532, 603: any of these parameters (1)`uint _value`, (2)`uint _tokens` and (3)`uint _rate` may be calculated from any two of them, so one of them is probably redundant.
26. Line 552, 621,685 : parameter `_token` is already listed as `ZMINE` type.
27. Line 553, 622: parameter `_rateContract` is already listed as `RateContract` type.
28. Line 554, 555 623: parameters `_whitelistPRE,_whitelistICO` and `whilelist` is already of type `WhiteList`.
29. Line 574, 646: `SafeMath` is used here as part of a business logic, and not as the second line of defense.
30. Line 648: `transfer` is more appropriate to use here. Also, it is inefficient to transfer every tranche to the owner separately from each other. Accumulating ether on

contract's balance and allowing the owner to withdraw accumulated ether all at once would be a better way.

31. Line 600: contract `PreSale` has much in common with `ICOSale` contract. Probably, it would be better to extract the common parts into an abstract base contract, and then inherit both, `ICOSale` and `PreSale` from it.

2.7 Major Flaws

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

1. Line 279: assignment `allowed[msg.sender][_spender] = 0` allows `_spender` to front run the transaction and spend allowance in such a way that `msg.sender` will not notice this.
2. Line 337: value for `holderIndex [owner]` is not set anywhere.
3. Line 461: variable `treasureBox[holder]` is available for the public modification.
4. Line 593 and 665: the functions does not fit into 2300 gas. This violates Solidity guidelines.

2.8 Moderate Issues

This section lists moderate issues which were found in the token smart contract.

1. Line 644: `div` may lead to rounding errors that may accumulate over time.
2. Line 704: sum of three values sent could be less than `_value` due to rounding errors.

2.9 Other Issues

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

1. Line 304: function `claim()` should be public.
2. Line 333: probably, perameter `_address` should be indexed.
3. Line 440, 444, 524: `isExchanger` and `airDropDestination` and `get` should be declared as public. So the functions `isExchanger`, `etAirdropDestination` and `getRate` will not be needed.
4. Line 496, 497, 498: `public` should be constant as well.
5. Line 543, 544: `startDate`, `stopDate` should be constant as well.
6. Line 546,547: `minTx`, `maxTx` should be constant as well.
7. Line 532, 533: parameter `address` should be indexed.
8. Line 612, 613, 614: `startDate`, `stopDate` and `maxTx` should be constant.
9. Line 615, 680: `minTx` should be constant.
10. Line 673: `_recipient` parameter should probably be indexed.

3. Our Recommendations

Based on our findings, we recommend the following:

1. Fix the most important flaws described in section 2.1.

2. Make the token EIP-20 compliant.
3. Fix arithmetic overflow issues.
4. Check the issues marked as “unclear behavior” against functional requirements.
5. Refactor the code to remove suboptimal parts.
6. Fix the documentation issues.
7. Fix the documentation, readability and other (minor) issues.