Hive Contracts: Audit Report

Mikhail Vladimirov and Dmitry Khovratovich

1st September, 2017

This document describes issues found in Hive sources during code review performed by ABDK Consulting.

1. Introduction

We were asked to audit a set of Hive contracts at the commit 0d54699. The set includes ERC20Interface.sol (interface contract for ERC-20 tokens), HVNToken.sol (a token contract), Migrations.sol, Owned.sol, SafeMath.sol, TokenRecipient.sol.

We got no additional documentation on the contracts so reviewed them as is.

2. ERC20Interface

This section is devoted to the analysis of the ERC20Interface.sol contract.

2.1 Minor Issues

To keep the interface clear, the `totalSupply` variable should be declared as a function (line 24).

3. Migrations

This section is devoted to the analysis of the MIgrations contract.

3.1 Documentation Issues

Documentation is not available for the contract purpose nor for its functions nor its variables (lines 19, 20, 21, 27, 31, 35). It is not clear what the contract functions should do.

3.2 Code Redundancy

The `restricted` modifier (line 23) can be replaced with the onlyOwner modifier from Owned.sol . The `owner` variable (line 20) can then be removed.

3.3 Missing Functionality

After multiple calls to `upgrade`, the value of `last_completed_migration` variable does not change. It probably should be increased (line 37).

3.4 Dangerous Behaviour

1. The `restricted` modifier (line 23) does not revert a transaction when a condition is not met and thus it is unsafe: if it is applied to a payable function, the received Ether might be pocketed.
2. There is no restriction on how `last_completed_migration` variable is modified (line 32). For example, it can be set to any previous value any time in the future. Maybe the `setCompleted` function should be made internal.

3.5 Critical Bug

The call to `setCompleted` function will probably fail when called in `upgrade` as long as the current contract is not the owner.

3.6 Minor Issues

1. The `last_completed_migration` and `new_address` variable is not named in camelCase like others (lines 35, 37).
2. In the `upgrade` function parameter type be `Migrations`, not `address`.

4. HVNToken

In this section we describe issues related to the sample token contract defined in HVNToken.sol.

4.1 ERC-20 Compliance Issues

The `approve` function throws if the current allowance is not 0, which contradicts the ERC-20 requirements and is thus incompatible with pre-existing token processing software.

4.2 Documentation Issues

1. The token contract has functionality beyond ERC-20 API so the comment that it is a standard ERC-20 contract is misleading (line 23).
2. No documentation comments for public fields (lines 27-34).
3. No documentation for `transfer, transferFrom, allowance, approveAndCall, mint, burn` inputs and return values.
4. No documentation for events (lines 193-196).

4.3 Arithmetic Overflow Issues

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

1. No overflow protection for addition (line 88, line 102).
2. In line 87, as well as in may other places in this smart contract, `SafeMath` is used not as the last line of protection, but as part of usual business logic. If this is desired usage, then `SafeMath` should do `require` of even `if (...) revert()` instead of `assert`.

4.4 Subefficient Code

This section lists subefficient code patterns found in token smart contract.

1. Prohibiting a zero address for token transfers (line 85) does not seem to solve any problem. Sending to zero address is common way to burn tokens, but not the only available way. If one will want to burn tokens, he may always bypass this protection by sending to, say, `0xdead`.

4.5 Redundant Functionality

1. The “short address” protection (line 39) looks redundant. There are too many ways to call a smart contract incorrectly (for example, confusing the order of parameters), of which the “short address” issue is the most known and thus usually fixed.
2. The `onlyOwner` modifier in the constructor (line 53) looks redundant because: (a) The constructor is always called by the owner of the smart contract, because whoever calls the constructor becomes an owner. (b) Function `mint` has `onlyOwner` modifier.
3. A parameter in `Freeze` and `Unfreeze` event is redundant since only owner can call the corresponding functions (line 65).
4. The `onlyPayloadSize` modifier makes little sense when applied to the constant functions (line 131) since when called by a human they (promise to) do not change the code, and calls via another contract are always properly formed.
5. One event should suffice in `mint` (line 155) and `burn` (line 169) .

4.6 Double Approve Issue

The ERC-20 API is vulnerable to the double-approve attack, where an allowance recipient is able to race the spender in order to benefit from two consecutive approvals. The contract protects from it by requiring the allowance to be 0 when approve function is called. As this breaks backward compatibility, we recommend introducing another approval function which would have a built-in protection. Such function can be defined as `approve(address _spender, uint _oldValue, uint _newValue )` taking assumed allowance value as the second argument. If actual allowance differs from an assumed one, this method just returns false.

4.7 Major Flaws

1. The `onlyPayloadSize` modifier for `approveAndCall` function would not work properly as the third parameter has variable length.
2. Remaining allowance not spent by `receiveApproval` is not revoked after call (line 144).

4.8 Other Issues

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

1. The `allowed` mapping (line 28) has two keys and semantics of the keys is not clear with comments.
2. Conditions on inputs should be enforced with `require` rather than `assert` (line 40, line 48) or simple checks (line 163) .
3. The issuance size of 500 mln should be made constant or a constructor parameter (line 55).
4. Not all branches of `approveAndCall` return value (line 144).
5. `ClaimTokens` method should probably have `ERC20Interface` type of its parameter (line 179) and internal variable (line 185).
6. The Transfer event (line 189) is confusing because it looks like somebody transferred HVN tokens from `_token` to `owner`, while actually some other kind of tokens was most probably transferred.

4.9 Test Issues

The project contains 15 test cases for automated testing. This is good but not sufficient, as not all functions are covered with tests. We recommend to add at least one test for every function.

- HVNToken.js lacks tests for Migrations.sol and TokenRecipient.sol, as well as for functions approveAndCall and claimTokens in HVNToken.sol.
- HVNToken.js, lines 29, 38, 46, 71, 80: discrepancies regarding zero-based vs one-based account enumeration. Account numbers in test descriptions are usually (though not always) one-based, while the accounts array starts with accounts[0]. Introduce a uniform enumeration of accounts to avoid ambiguity.

5. Owned

Owned.sol is a smart contract that handles ownership modifiers.

5.1 Documentation Issues

1. Copyright and license information are missing.
2. Contract, functions, public fields, modifiers lack documentation.

5.2 Other Issues

1. `newOwner` variable may not need to be public as long as it is not used by other contracts or ordinary users (line 5).
2. The input constraints are typically enforced with `require` rather than `assert` (line 12).
3. `OwnerUpdate` name is confusing (better `OwnerChange`).

6. SafeMath

6.1 Documentation Issues

1. Add “and divide” to the comment (line 17).
2. Replace “x+y” with “a+b” (line 26).

6.2 Arithmetic Overflows

1. The functionality of `add`, `mul` relies on unspecified overflow behavior in the addition and multiplication. It is better to check for overflow before adding/multiplying (lines 29, 55).
2. The functionality of `div` relies on division by zero behavior that is different in different versions of Solidity and could change again in future. It is better to check for dividing by zero before actually dividing (line 68).

6.3 Other Issues

Variable `c` not needed (line 68).

7. TokenRecipient

7.1 Documentation Issues

1. Contract, functions, inputs lack documentation.
2. The `_to` parameter is confusing and needs documenting.
3. Comment is useless (line 18).

8. Security provisions

We recommend the smart contract designers to claim explicitly security provisions in addition to the intended functionality of the contract. Such provisions should be non-trivial falsifiable claims, i.e. they should declare certain property of the contract for which an attack can be

demonstrated if implemented incorrectly. The security provisions serve not only for the purpose of confidence of future users in the contract's behaviour, but also as a platform for a potential bug bounty program. The contract authors are encouraged to offer various bounties binded to some provision being violated.

Here we provide a list of security provisions which are appropriate for these contracts and would be expected by ordinary users.

Name	Description	Current status
HVNToken		
Constant supply.	Sum of all positive balances is constant between mintings	TRUE
Safe multiple approvals.	A token holder can approve another address with some tokens only if the previous allowance has been fully spent.	TRUE
No lost Ether or Token	Every token or Ether that is sent to the contract is either reverted or can be reclaimed later by the contract owner.	TRUE

9. Our Recommendations

Based on our findings, we recommend the following:

1. Fix the setCompleted issue;
2. Fix the approveAndCall issue;
3. Fix overflow issues;
4. Refactor the code, remove redundant parts, optimize subefficient parts.