Wallet
Smart Contract. Audit

Mikhail Vladimirov and Dmitry Khovratovich

3d September 2019

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

1. Introduction

We've been asked to review the Wallet smart contract given in a public gist file.

2. Wallet

In this section we describe issues related to the smart contract defined in the Wallet.sol. In general, we find that both supply and withdraw scenarios contain a lot of mistakes and are difficult to be executed correctly by a user. There are many ways to make the transaction fail because of many checks, but on the other hand several checks can be bypassed.

We recommend a significant refactoring of the code, removing unnecessary checks, and splitting some functions into many. We also recommend removing the closeAccount feature, as it is non-trivial to implement it correctly for token-handling contracts.

We finally note that the C-style error reporting (0 for success) is uncommon for Ethereum. Common patters are:
1. Return true on success, false on error
2. Return on success, throw (optionally with error message) on error

2.1 Critical Flaws

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

1. Line 128: there is no check that this is a real `cToken` corresponding to the `address _tokenAddress`. A user may submit here any contract address with valid `balanceOF` and `redeem` methods and withdraw any token without extra checks or fee payment, including `cTokens`. In case redeeming all

`cTokens` will produce more than `principalSupplied` tokens, this call to `withdrawFromMoneyMarket` will fail. This may happen in case `cTokens` were sent directly to the contract.

2. Line 141: the `updatePrincipalSupplied` will decrease `principalSupplied` by the whole amount of `_Tokens` currently sitting at contracts account. Apart from recently redeemed tokens, this amount may include some `_Tokens`, that were deposited but not yet converted to `_CTokens`, thus not yet added to `principalSupplied`. The user will lose tokens in this case, because `principalSupplied` will be charged for tokens that were never added to it. To fix this either ensure that current `_Tokens` balance of the contract is zero, or calculate the exact amount of redeemed tokens and use this value inside `updatePrincipalSupplied` instead of contract's `_Tokens` balance.

2.2 Major Flaws

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

1. Line 137: the operation `_cTokensRequested = _totalCTokens` indicate that the method may actually withdraw less tokens than requested, and still return successfully. This could be dangerous.
2. Line 139: the check `_CToken.redeem(_cTokensRequested) == 0` reverts the entire transaction, which might limit the availability of other tokens, for example if `closeAccount` is called. Consider handling the error differently.
3. Line 139: according to the documentation, the `redeem` returns zero on success and error code on fail. The returned error code is lost in this line. It makes harder to investigate problems. To fix this, either the `redeem` function should throw with an error message on fail, or `withdrawFromMoneyMarket` should return error code instead of throwing (we recommend the former).
4. Line 144: returned value is ignored for the transfer.
5. Line 145: if the entire amount can not be withdrawn for some reason, the `assert()` function throws and the entire transaction reverts. It might make sense to retrieve a predefined amount of tokens.
6. Line 165: the function `closeAccount()` may fail for multiple reasons, including balance checks and overflows in `withdrawERC20`, `withdrawFromMoneyMarket`, or in the functions it calls. Consider removing unneeded exceptions and introducing non-throwing overflow checks.
7. Line 189: after performing the `selfdestruct(address(this))` function the user will lose all tokens on contract's balance other than DAI, USDC, cDAI, and cUSDC, that could otherwise be recovered via withdrawERC20 function.

2.3 Moderate Flaws

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

1. Line 62: the description "`Approves the token contract to send a large amount of tokens to the cToken contract`" is misleading. The function actually approves `_cTokenAddress` to take any number of `_Token's` from this contract.

2.4 Documentation Issues

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

1. Line 26: the events `GossamerContractCreate`, `GossamerApproval`,`GossamerSupply`, `GossamerWithdrawal`, `GossamerAccountClose` and their fields are not documented.
2. Line 26: usually events named using nouns rather than verbs,e.g. `GossamerContractCreation`.
3. Line 43,143: these lines probably should be deleted.
4. Line 104, 114, 157: typo in these lines. There should be `equation` instead of `equaton`, `arise` instead of `arrise` and `use` instead of `us`.

2.5 Arithmetic Overflow Issues

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

Line 113: the multiple operation may be cause an overflow (even if final result fits into uint256).

2.6 Suboptimal Code

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

1. Line 18-21: the addresses `daiAddress`, `usdcAddress`, `cDAIAddress` and `cUSDCAddress` are valid for the Rinkeby network. One will have to change them before deploying on Mainnet. These addresses are not declared constant, which mean that they do occupy storage space and are more expensive to access. We are recommend to replace these storage variables with public getter functions like this:

```
function daiAddress () public returns (address) {
     return 0x89d24A6b4CcB1B6fAA2625fE562bDD9a23260359;
// Mainnet address
}
etc.
```

Then, for testing on Rinkeby, derive another contract from this one and override function in it. This way you will not need last minute changes, your code will consume less gas, and public API will be preserved.

2. Line 26,27,28: the parameters in these lines should probably be indexed.
3. Line 67: surrounding brackets in this line are redundant.
4. Line 79: passing sign bit as separate boolean parameter is unoptimal. Consider using signed type for the `_interestEarnedOnWithdrawalAmount` parameter or just split this method into two separate methods (we are recommend the latter).
5. Line 97: the method `_contractBalance` always tries to mint `cTokens` for the full balance of tokens. It could be convenient to implement an ability to mint `cTokens` only for part of the balance.
6. Line 98: the check `_Token.balanceOf(address(this)) == 0 && _CToken.balanceOf(address(this)) > 0` seems to be useless. It does not guarantee that `cTokens` on the contracts balance are those just minted. Consider removing it or check that balance of `cTokens` actually increased after minting.
7. Line 114: the comment "`Need this check for rounding errors that can arrise with low decimal tokens (e.g., USDC)`" is incorrect. The `mul` operation does not round, and the `div` always rounds down, so rounded value cannot be greater than precise value.
8. Line 126: it would be better to log the events in the caller.
9. Line 130: the variable `_CToken` (and other similar ones) is redundant.
10. Line 132: the check `_CToken.balanceOf(address(this)) > 0` makes every caller who does not want an exception to check the balance himself, which is extra work.
11. Line 140: the parameters `_tokenAddress, _Token` always have the same value. Consider removing one of them.
12. Line 144: transferring the whole balance could fail for token contracts that charge transfer fee. Consider allowing user to specify withdrawal amount. We also recommend to split this method into two methods: one to convert `CTokens` to Tokens on contracts balance, and another to withdraw Tokens from contract's balance to user's own address.
13. Line 150: function as `function () external payable {` is not needed since Solidity 0.4.0, as without it all Ether will be reverted anyway. Also, this `payable` modifier is misleading, because it explicitly says that the function accepts ether transfers, while the main purpose of the function is to reject ether.
14. Line 156: ReentrancyGuard documentation recommends to make functions as `onlyUserAndAdmins` external. Also, there is no need to make it `nonReentrant`. It does not modify the contract state.
15. Line 160: the check `_tokenBalance > 0` makes code heavier.
16. Line 177: the `balanceOf` is computed two times just because `withdrawFromMoneyMarket` throws on zero amounts. This costs extra gas, consider optimization.

2.7 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 37: the comment is unclear: addresses are identical or functionality is identical? In the latter case, there should be a more comprehensive naming for variables.
2. Line 41: the purpose of the fixed number of admins is unclear.
3. Line 83: if someone sends `_Token's` to this contract then the value of the `principalSupplied[_tokenAddress]` will be smaller than the contracts balance, the underflow will throw an exception, and `CloseAccount` method will fail. Is this on purpose?

2.8 Other Issues

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

1. Line 28: perhaps, the event should be `GossamerDeposit`, not `GossamerSupply`.
2. Line 29: perhaps, the event should be `GossamerAccountClosing`, not `GossamerAccountClose`.
3. Line 65,91,128: the `address` should be ERC20Interface rather than just `address`.

3. Summary

Based on our findings, we also recommend the following:

1. Fix critical issues which could lead to asset loss.
2. Pay attention to major and moderate issues.
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.