TokenCard Contract Audit - DRAFT

February, 2018

New Alchemy

Introduction

During February of 2018, Token engaged New Alchemy to audit smart contracts that they created to support the TokenCard platform.

The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts.

Token provided New Alchemy with access to their code repository and whitepaper.

Files Audited

New Alchemy reviewed the Solidity code (.sol files) in the GitHub repository https://github.com/tokencard/tokencard at commit hash `44574669a3240afe2b4f8bd16b354d292dd409aa`.

One additional file under review was not part of the repository, but instead in Etherscan at address 0xaaaf91d9b90df800df4f55c205fd6989c977e73a.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bugfree status. The audit documentation is for discussion purposes only.

Executive Summary

The code largely followed best practices including:

- two-phase ownership transfer
- short address attack mitigations
- double-spend (via `approve`) attack mitigations
- mechanisms to pause token operations and stop minting new tokens
- checks for arithmetic overflows

However, some flaws were still found. Not all contracts implement two-phase ownership transfer and not all applicable functions mitigate short-address attacks. Further, one function appears to be inoperable due to a deficient short-address attack mitigation.

General Discussion

The `Token` contract, being already deployed since May 2017, was written for a rather old version of Solidity. Any revisions to this contract should bring it up to date with the latest guidelines for Solidity, including the following:

- The version pragma specifies a version (0.4.4) that is not the latest version of Solidity at the time of the audit, which is 0.4.20. Use the latest version of Solidity in order to take advantage of the latest features and improvements that Solidity has to offer.
- Use the convenience functions `assert` and `require` instead of the deprecated `throw`. They were added in Solidity 0.4.10.
- Explicitly set function visibility instead of using the implicit default.

While `Wallet` and `Controller` implement functionality to change owners, there is no method for the controller of a wallet to be changed. consequently, there is no way to assign an existing wallet to an upgraded controller. It is unclear if this is intentional or not. If not, then consider adding logic to `Wallet` to implement a two-phase transfer of the controller, possibly requiring approval from the wallet's owner as well.

Moderate Issues

`Token.approveAndCall` always fails

The function `Token.approve` uses the modifier `onlyPayloadSize(2)`, which asserts that `msg.data.length` is equal to 68, presumably as a mitigation against short-address attacks. This function is `public` (it's not explicitly specificed, but `public` is the default visibility in Solidity), so it can be called both internally and externally. However, `msg.data` is not re-written on internal calls: it continues to hold the call data from the original external call. Hence, `msg.data` may have a different length if this function is called internally.

`Token.approveAndCall` makes an internal call to `approve`. Further, its parameters are an `address`, a `uint256`, and a `uint[]`. Since even an empty array has non-zero size in memory, an external call to this function will have `msg.data.length` greater than 68. This will result in the call to `approve` always failing, preventing this function from ever working.

The best way to fix this issue without removing the short-address attack protections is to create two versions of `approve` (and of any other functions that require short-address attack protection and may be called internally): one `internal` function that does not check `msg.data.length`, and one `external` function that does check `msg.data.length` then calls the internal function. More details on short-address attack mitigations are given below.

Minor Issues

Possible Integer Overflow

The addition operation on line 109 in `Token.totalSupply` could overflow if the operands are too large. Use `SafeMath.add` instead.

Missing of short-address attack protections

Some Ethereum clients may create malformed messages if a user is persuaded to call a method on a contract with an address that is not a full 20 bytes long. In such a "short-address attack", an attacker generates an address whose last byte is 0x00, then sends the first 19 bytes of that address to a victim. When the victim makes a contract method call, it appends the 19-byte address to `msg.data` followed by a value. Since the high-order byte of the value is almost certainly 0x00, reading 20 bytes from the expected location of the address in `msg.data` will result in the correct address. However, the value is then left-shifted by one byte, effectively multiplying it by 256 and potentially causing the victim to transfer a much larger number of tokens than intended. `msg.data` will be one byte shorter than expected, but due to how the EVM works, reads past its end will just return 0x00.

This attack effects methods that transfer tokens to destination addresses, where the method parameters include a destination address followed immediately by a value. In the Token contracts, such methods include `Token.approveAndCall`, `Controller.transfer`, `Wallet.debit`, `Wallet.hold`, `Wallet.release`, `Wallet.settle`, `Wallet.withdraw`, and `Wallet.transfer`.

While the root cause of this flaw is buggy serializers and how the EVM works, it can be easily mitigated in contracts. When called externally, an affected method should verify that `msg.data.length` is *at least* the minimum length of the method's expected arguments (for instance, `msg.data.length` for an external call to `Token.transfer` should be at least 68: 4 for the hash, 32 for the address (including 12 bytes of padding), and 32 for the value; some clients may add additional padding to the end). This can be implemented in a modifier. External calls can be detected in the following ways:

- Compare the first four bytes of `msg.data` against the method hash. If they don't match, then the call is internal and no short-address check is necessary.
- Avoid creating `public` methods that may be subject to short-address attacks; instead create only `external` methods that check for short addresses as described above. `public` methods can be simulated by having the external methods call `private` or `internal` methods that perform the actual operations and that do not need to check for short-address attacks.

In the `Token` contract, the functions `mint`, `transfer`, `transferFrom`, `approve`, `increaseApproval`, and `decreaseApproval` all protect against short-address attacks by unconditionally checking that `msg.data.length` is equal to an expected value. This approach may cause compatibility problems with some client software that adds additional padding to `msg.data`. Further, since all of these functions are public, anything that calls them internally will fail if the original external call had a different call data length, as is the case with `approveAndCall`'s internal call to `approve`. Consequently, these functions' protections should be improved as well.

Whether or not it is appropriate for contracts to mitigate the short-address attack is a contentious issue among smart-contract developers. Many, including those behind the OpenZeppelin project, have explicitly chosen not to do so. While it is New Alchemy's position that there is value in protecting users by incorporating low-cost mitigations into likely target functions, Token would not stand out from the community if they also choose not to do so.

Lack of two-phase ownership transfer

While `Token` and `Controller` implement two-phase ownership transfer, `Wallet` does not. Consequently, a mistake in changing the ownership of this contract could result in it becoming irrecoverably unowned.

Line by line comments

This section lists comments on design decisions and code quality made by New Alchemy during the review. They are not known to represent security flaws.

Token.sol

Line 1

Solidity 0.4.4 is very old. Any new deployment of this contract should use a more recent version; the latest at this time is 0.4.20.

Line 6

The state variable `DECIMALS` should have a `constant` modifier. This prevents assignment and does not take up storage space.

Line 20: `Owned.changeOwner`

Consider adding a check for transfer to the zero address. This can catch calls to functions where parameters may not have been initialized by the caller.

Consider adding a `OwnershipTransferred` event as is done by the OpenZeppelin project in contract `Ownable`.

Line 50: `SafeMath.assert`

The function shadows a builtin function of the same name. The builtin function was added in Solidity 0.4.10. Remove the function and use the builtin one.

Line 98: `Token.Launch`

To improve code clarity, follow the Solidity recommendation of using *mixedCase* notation for function names. Otherwise, this function looks like an event.

Line 170: `Token.multiMint`

To save gas, consider using a constant for `D160-1`. The unchanging value is calculated every iteration of the loop.

Line 205: `Token.transferFrom`

Explicitly set variable type in variable initialization. Use of the `var` keyword has been deprecated.

Line 297: `Token.logTokenTransfer`

To improve code clarity, follow the Solidity recommendation of using *CamelCase* notation for event names. Otherwise, this event looks like a function.

Additionally, consider using `indexed` parameters in events so that clients will be able to filter on desired topics, instead of manually having to go through the log data for a contract.

controller.sol

Lines 21-25

Consider using an enum rather than multiple integer constants.

Line 63: `process`

As written, this function uses the "arrject" pattern for its parameters. This pattern is bug-prone as anything calling this function must ensure that the arrays are all of the same length and that the data in them is correctly correlated. This function could be made simpler and safer by creating a struct containing an operation code, wallet address, token address, and amount, then passing an array of such structs to this function.

wallet.sol

Line 52

A comment describing the significance of the magic number 80000 would be helpful.

Lines 78, 91, 92, 102, 110, 120, 134, 137

It is more gas-efficient to only retrieve state variables only once. Each of the listed locations will read `overdraft[token]` again despite it already having been read in the current function.

Lines 91, 102

Rather than implementing explicit checks for overflow in arithmetic expressions, use SafeMath to perform arithmetic. This approach guarantees that the check matches the arithmetic actually performed.

Lines 101, 108: `hold`, `release`

No distinction is made between overdrafts due to holds and overdrafts due to excessive debits. Consequently, a call to `release` may remove an overdraft created by excessive debits, and a call to `settle` may remove a hold. Is this intentional? If not, holds and overdrafts should be maintained separately.

Line 136

The check if `amount > 0` is unnecessary; this condition was already established at line 123.