Token Group Ltd. Partial Audit

July, 2019

Contents

Chapter 1

Introduction

1.1 Scope of Work

Token Group Ltd. requested technical feedback on modifications to the Token Group Ltd. Wallet, a collection of smart contracts implemented in the Solidity programming language.

This review covered a subset of the source code present in the GitHub repository at `https://github.com/TokenCard/contracts` at the following git commit:

`94526e97715040a1bd53515582d031a960bc3e27`

Token Group Ltd. recently completed a full source-code audit of the following git commit:

`b99b7d1670f9ad7b90335e8391fe63fd7e20de9b`

There are 175 commits (including merge commits) in the above commit range.

The focus of this audit was to identify new issues introduced between the old and new source code revisions, and to identify any issues missed by the most recent Trail of Bits audit. This audit *does not* reiterate issues discussed in the most recent audit of the source, nor does it reiterate issues uncovered by the last audit of this source tree performed by this auditor. Additionally, external code dependencies (including in-tree external dependencies) were not within the scope of this audit.

This source code review, like any source code review, is not an endorsement of the code or its suitability for any legal/regulatory regime, and it is not a definitive or exhaustive list of defects.

Chapter 2

Summary

This audit uncovered one major and several minor defects in the source code.

2.1 Architecture

2.1.1 Trusted Parties

The TokenCard wallet spending limit functionality requires an exchange rate oracle in order for the spending limit to be denominated as a single unit, rather than a vector of units. Presently, the cryptocompare.com API and oraclize.it provide this functionality in the code. Consequently, both CryptoCompare and Oraclize are trusted parties insofar as the spending limit functionality is concerned.

The Token Group Ltd. is a trusted party for wallet whitelists, `holder` disbursement rates, and indirectly for token exchange rates for calculating spending limits. (The Token Group Ltd. controls the ENS resolver used to determine the exchange rate oracle.)

2.1.2 Threat Model and Trusted Parties

The Token Group Ltd. should take careful note of the trust chain responsible for delivering exchange rates. Specifically, if Oraclize "trusts" responses from cryptocompare.com over HTTPS, then in practice they trust the certificate chain beginning with a multi-domain certificate owned by Cloudflare, and *not* by cryptocompare.com. Practically speaking, this means that a compromise of cryptocompare.com's Cloudflare credentials is sufficient to compromise the exchange rate trust chain. Credential compromise is not a theoretcial concern: the `info@cryptocompare.com` email address was involved in the Intellimost data exposure in March of 2019.

You can use `haveibeenpwned.com` to search for email addresses that have been involved in widely-reported breaches.

In the event that admin credentials are compromized at Oraclized or CryptoCompare, the wallet spending limit functionality could cease to function correctly. Additionally, `Wallet.loadTokenCard()` may not work as intended, as exchange rates influence card-loading.

In short, The Token Group Ltd. should analyze the security posture of its partners on an ongiong basis and make architectural changes as necessary, since those partners influence security-critical behavior of the TokenCard wallet contract.

Chapter 3

Major Defects

The following defects are considered major defects in that they represent significant deviations between the expected or documented behavior of the code and its behavior in real-world conditions such that an attacker can exploit those deviations.

3.1 Wallet: Whitelist and Spending Limit Bypass

In the event of an end-user key disclosure, an attacker can bypass the wallet's address whitelist through the `executeTransaciton{}` function.

The `executeTransaction()` function blacklists the ERC20 `transfer()` and `approve()` methods insofar as it continues to enforce spending limits and whitelist rules on for those methods. However, those are not the only methods that ERC20 tokens can implement for transferring tokens. For example, many tokens implement methods like `increaseApproval()` and `decreaseApproval()`, which would still allow an attacker to create an allowance for an attacker-controlled account without respect to whitelist and spending limit rules.

Since many ERC20 tokens have mutable APIs (i.e. they use upgradeable contracts), it it may not be possible to guarantee that the `executeTransaction()` functionality is ever completely safe, even if it were to incorporate a much larger blacklist of functions.

Chapter 4

Minor Defects

The following defects are considered minor defects in that they are not exploitable without elevated privileges, or because there are no obvious code paths that exist to trigger these bugs. However, readers should note that minor defects can become major defects as a consequence of plausibly-benign code changes.

4.1 BytesUtil: Bounds Check Bypass through Arithmetic Overflow

In the functions `_bytesToAddress()`, `_bytesToUint32()`, and `_bytesToUint256()` in `bytesUtil.sol`, the explicit bounds check at the beginning of the function can be bypassed through arithmetic overflow.

For example, consider:

```
require(_bts.length >= _from + 20);
```

If `_from + 20` overflows and wraps to an integer less than `_bts.length`, then the subsequent inline assembly will load memory beyond the bounds of the array.

Although this issue is not presently exploitable (because the affected functions are all called with constant arguments), the bounds checks should still be patched to handle arithmetic overflow in case a future patch passes an attacker-controlled `_from` parameter to one of these functions.

4.2 Whitelist: Unbounded Gas Use

Removing a token from `tokenAddressArray` has a worst-case gas cost that scales linearly with the number of tokens in the array. Consequently, it is possible for `addTo-`

kens() to add enough tokens that removeTokens() cannot execute within the block gas limit.

One possible mitigation is to require the caller of removeTokens() to provide the index of the token(s) to be removed in tokenAddressArray, and have the contract check that the indices are valid before removing the tokens.

4.3 Base64: Mishandling of Malformed Input

The _base64decode() function in base64.sol silently decodes null bytes when the input uses characters outside of the base64 alphabet.

In general, functions that handle untrusted data should eagerly detect and report malformed input.

4.4 Other Notes

- The strings library contains a large amount of unchecked arithmetic and inline assembly. It is reasonbly likely that there are bugs lurking in this code, some of which may be exploitable due to the use of inline assembly.
- The Oraclize contract duplicates some of the string-manipulation logic in the strings library (see Oraclize.strCompare() and Oraclize.indexOf()).
- In walletDeployer.sol, the deployWallet() and migrateWallet() share identical code on lines 62-66 and 78-83. Consider refactoring out these common lines into a private function.
- In tokenWhitelist.sol on line 79, the variable _stabelcoinAddress is misspelled.
- In wallet.sol, the comment on line 727 refers to the variable _amount, when it actually applies to the variable amountToSend.
- In base64.sol, lines 26 through 54 are improperly indented.

4.5 Selected Unresolved Issues

The following is a selected list of issues reported in previous audits that remain unresolved. This is not an exhaustive list of unresolved issues.

- The signature scheme used by Oraclize does not incorporate ECDSA signature malleability countermeasures. Additionally, the verifyProof() function parses untrusted input before signature validation.
- The source tree includes compiler-generated artifacts. Production artifcats should not be checked in, and they should only be generated by a trusted build machine.

- The exchange-rate scheme used by the wallet incorporates an unnecessarily large number of trusted parties, to the detriment of the security of the end-user. In this scheme, as in most forseeable schemes, Token Group Ltd. is a trusted party. Additional trusted parties increase the wallet's attack surface.