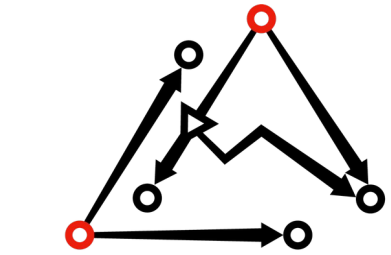

Least Authority

PRIVACY MATTERS

TzBTC

Security Audit Report

Tezos Foundation

Final Report Version: 13 March 2020

Table of Contents

Overview

Background

Tezos Foundation has requested that Least Authority perform a security audit of the following:

- **TzBTC** (BTC-backed token on Tezos): Enables the compliant issuance of a fully Bitcoin-backed token on the Tezos blockchain while aiming to eradicate the risks of a single-point-of-failure. This is achieved by dividing the various tasks into keyholders that are responsible for the custody of BTC and the issuance of respective TzBTC, based on an m-out-of-n multi-signature setup and regulated financial intermediaries that act as gatekeepers to ensure compliance with money-laundering and terrorist financing legislation.

Project Dates

- **November 25 - December 18**: Initial Review *(Completed)*
- **January 3**: Initial Audit Report delivered *(Completed)*
- **February 24-27:** Verification Review *(Completed)*
- **February 28:** Final Audit Report delivered *(Completed)*
- **March 13:** Updated Final Audit Report delivered *(Completed)*

Review Team

- Ramakrishnan Muthukrishnan, Security Researcher and Engineer
- Sajith Sasidharan, Security Researcher and Engineer
- Mirco Richter, Cryptography Researcher and Engineer
- Nathan Ginnever, Security Researcher and Engineer
- Emery Hall, Security Researcher and Engineer
- Alex Leitner, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of TzBTC followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following code repositories are considered in-scope for the review:

- **TzBTC:** Tezos Foundation shared `tezos-btc.tar.gz` with Least Authority on 7 November 2019

For the TzBTC Contract, we examined the version contained within `tezos-btc.tar.gz` with a SHA256SUM:

`af676f6ec4df9e6898533f01b8eac470cfdf3348397338a73722cfc6c398951f`

For our follow up verification , we examined the version contained within `tezos-btc-3a785c4.zip` with a SHA256SUM:

`8f920771e4d11db5773999e29927c44bed6a36ba21999bd617b32019d7af4950`

All file references in this document use Unix-style paths relative to the project's root directory.

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the network;
- Potential misuse and gaming of the smart contracts;
- Attacks that impacts funds, such as the draining or the manipulation of funds;
- Mismanagement of funds via transactions;
- Economic incentives: ensure token economics (monetary incentives to punish bad behavior and reward good behavior) are included and functional;
- DoS/security exploits that would impact the contracts intended use or disrupt the execution of the contract;
- Vulnerabilities in the smart contracts code;
- Protection against malicious attacks and other ways to exploit contracts;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

As part of our investigation and analysis, we built the Haskell code that generated two executables: `tzbtc` and `tzbtc-client` followed by deploying the code using `tzbtc-client` that originated the version V0 of the contract and upgraded it to V1. Our review of the Lorentz code included looking at each entry point, running tests, reviewing the upgrade mechanism for vulnerabilities, and examining methods to get the upgraded contract from the original contract and deserialize it.

A considerable amount of the review time was applied to surveying how upgrades work and what can be modified (code, parameters, number of parameters in an upgraded entrypoint, types of the parameters in the upgraded code, etc.) and what safety guarantees are or are not provided during the upgrade.

Although the use of a statically typed language, Haskell (with many dependent typing features turned on), and the target typed language, Michelson, helps to arrive at correct programs, we recommend the use of formal verification tools. It would be useful to derive properties from these contracts and add them in the form of property tests.

We found the Michelson code difficult to read because of its low level nature. There are certain useful idioms and patterns that become apparent only after one has read a substantial portion of the code. Some of them seem to be captured by the Morley's Macro module. Even with that, we found Lorentz code difficult to understand.

We spent a significant portion of time asking questions that could be addressed with the availability of better documentation. We recommend that additional documentation be written to incorporate answers to common questions. The Michelson web page does not document common idioms. However, the 4-part Michelson tutorial was very helpful in becoming familiar with the other tools.

Specific Issues

We list the issues we found in the code in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Lack of Generic Multisig Tests	Resolved
Issue B: Generic Multisig May Update Owner Set Improperly	Resolved
Issue C: Multisig Parameters Could Be Improperly Constructed	Resolved
Suggestion 1: Improve Documentation	Resolved
Suggestion 2: Consider Protecting Against an Accidental Update to a Malicious Valid Address	Unresolved

Issue A: Lack of Generic Multisig Tests

Location

/tezos-btc/contracts/MultisigGeneric.tz

/tezos-btc/src/Lorentz/Contracts/TZBTC/MultiSig.hs

Synopsis

Tests that confirm the intended functionality of the generic multisig contract were not present in the code. Since there are no tests on the multisig contract itself, it is assumed that tests exist elsewhere. It is difficult to understand the functionality of the multisig contract without proper tests.

Impact

The multisig contract will be the controlling address of the token and unexpected behavior can cause loss of control of the token.

Mitigation

We suggest writing proper tests on the multisig contract.

Status

Multisig tests for various cases have been added, located at test/Test/MultiSig.hs . There are a few incorrect comments due to copy/paste from a previous test ("Use Alice's public key but bob's secret.") at Line 168, 206, 246.

Verification

Resolved.

Issue B: Generic Multisig May Update Owner Set Improperly

Location

/tezos-btc/contracts/MultisigGeneric.tz

Synopsis

Change_keys allows for changing control of the multisig contract by changing the owner keys. The /tzbtc/README.md points to a multisig contract with functionality to update the signing accounts existing on the multisig where there could be issues with locking the contract's ownership out at the multisig level.

Impact

Control will be lost if there is not a proper modifier to limit the ability of ownership change to prevent the case where there are no owners of the multisig.

Preconditions

An improper call to update the owner set to a null set or a set below the threshold of necessary signers to execute a block of code. It appears that the update function simply replaces state with the incoming transaction. If a null set of keys is updated to that state the contract and everything it owns will render useless.

Feasibility

This mistake could happen easily if no checks are provided.

Remediation

Consider adding checks to the change_keys function that will not allow for a null set or lower than required threshold set of owners to be introduced.

Status

A new command called deployMultisigContract has been added and a corresponding eponymous function in src/Client/IO.hs checks whether the passed threshold is a non-zero value and is lower than or equal to the list of keys.

Verification

Resolved.

Issue C: Multisig Parameters Could Be Improperly Constructed

Location

/tezos-btc/contracts/MultisigGeneric.tz

/tezos-btc/src/Lorentz/Contracts/TZBTC/MultiSig.hs

Synopsis

We did not identify any tests for the multisig wallet that could be used to control sensitive operations on the token. If a generic multisig wallet is used, then signers may unknowingly process an unexpected transaction.

Impact

The parameters may be improperly constructed before signing which could lead to signatures on transactions that were not intended (i.e. an improper update address). If the multisig contract owns the token, this could cause total loss of control.

Feasibility

The client provides packages and documentation about ways to execute human readable multisig transactions for each multisig function that the general contract wraps so that the likelihood of this mistake happening is reduced.

Technical Details

The contract allows for the "generic multisig" contract to control execution of generic functions on the contract system. The inputs to this multisig signing will be Michelson code snippets that are difficult to read. We did not identify any client side tools that will help the signers understand the code that is being proposed to the multisig wallet.

Mitigation

Consider using a multisig wrapper contract that only allows signing specific parameters.

Status

The generic multisig contract that allowed signers to act as a blockchain entity, executing arbitrary code, has been replaced with a specific multisig contract that only executes what the wrapped contract expects.

Verification

Resolved.

Suggestions

Suggestion 1: Improve Documentation

Location

README.md

Synopsis

We found there to be a lack of documentation for common idioms, which is not included on the Michelson web page.

Mitigation

We recommend including documentation that would allow new contributors and reviewers to understand the entire system more easily and efficiently.

Status

A considerable amount of information on using the tezos-client has been added to the README.md. The generated documentation (using the command "tzbtc printContractDoc ") has additional information and pointers on upgradeable contracts. The Morley upgradeable contracts documentation has also been updated.

Verification

Resolved.

Suggestion 2: Consider Protecting Against an Accidental Update to a Malicious Valid Address

Location

Lines 200 and 216 in:

src/Lorentz/Contracts/TZBTC/Impl.hs

Synopsis

The current two step process to update the ownership of the contract is not a full solution to improper format in an owner update in the case of an accidental update to a malicious, but valid, address.

Mitigation

We recommend investigating alternative protections against this. An accidental update is only feasible if the owner account is compromised such that the attacker can set the new owner to themselves.

Status

Tezos states that the current two step procedure is sufficient. We recommend that our suggested mitigation of investigating alternative protections be reconsidered.

Verification

Unresolved.

Recommendations

We recommend addressing the errors noted in the Status section of Issue A and reconsider the use of alternative protections in Suggestion 2.

Otherwise, we commend Tezos for addressing the *Issues and Suggestions* in this report and recommend that future development releases continue to apply security best practices and that additional security audits be conducted to address any potential issues and vulnerabilities.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. Although we are a small team, we believe that we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create

an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.