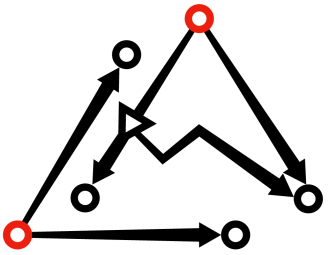

Least Authority

PRIVACY MATTERS

Atomex Smart Contracts

Security Audit Report

Tezos Foundation

Final Report Version: 6 May 2020

Table of Contents

Overview

Background

Tezos Foundation has requested that Least Authority perform a security audit of the Atomex Smart Contracts.

- **Atomic Swap Contract (Tezos):**
  - Smart contract implemented using Morley library framework and developed in a Haskell based domain specific language that compiles to Michelson
- **FA1.2 Contract:**
  - Smart contract implemented in PascaLIGO
- **Atomic Swap Contract (Ethereum):**
  - Smart contract implemented in Solidity
- **Atomic ERC-20 Contract:**
  - Smart contract implemented in Solidity

Project Dates

Atomic Swap Contract Tezos + FA1.2

- **April 8 - April 17:** Code review *(Completed)*
- **April 20:** Delivery of both Initial Audit Reports *(Completed)*
- **May 4 - 5:** Verification *(Completed)*
- **May 6:** Delivery of both Final Audit Reports *(Completed)*

Atomic Swap Contract Ethereum + Atomic ERC-20 Contract

- **April 8 - April 14:** Code review *(Completed)*
- **April 20:** Delivery of both Initial Audit Reports *(Completed)*
- **May 4 - 5:** Verification *(Completed)*
- **May 6:** Delivery of both Final Audit Reports *(Completed)*

Review Team

- Mirco Richter, Cryptography Researcher and Engineer
- Nathan Ginnever, Security Researcher and Engineer
- Phoebe Jenkins, Security Researcher and Engineer
- Dylan Lott, Security Researcher and Engineer

Coverage

Target Code and Revision

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

- **Atomic Swap Contract (Tezos):** https://github.com/atomex-me/atomex-michelson
- **FA1.2 Contract:** https://github.com/atomex-me/atomex-fa12-ligo
- **Atomic Swap Contract (Ethereum):** https://github.com/atomex-me/atomex-solidity
- **Atomic ERC-20 Contract:** https://github.com/atomex-me/atomex-erc20-solidity

Specifically, we examined the Git revisions for our initial review:

Atomic Swap Contract (Tezos-Michelson): `f36c4d600cc0fd0c942f789dc0a0cdd6b1caa885`

FA1.2 Contract: `142c29346d010301dcb1f930d46ccd0d98a462b3`

Atomic Swap Contract (Ethereum): `e5d4c03b4bcd735a0cb456cd99ec6d68cfbb98de`

Atomic ERC-20 Contract: `fe568c33dfffce6c495faf71af8a11ce6dad00f1`

For the verification, we examined the Git revision:

Atomic Swap Contract (Tezos-Michelson): `82f452d1ea4d0263b7a4eaab782a6e02b06bcaf3`

FA1.2 Contract: `6e093b484d5cf1ddf66245a6eb9d8d11dfbb45da`

Atomic Swap Contract (Ethereum): `cc8b7f56223290985083475 68bb1854c121c93c2`

Atomic ERC-20 Contract: `3a6e1cefd477cce067437d6b222b05e5ee3d9af1`

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

Areas of Concern

All Contracts

The following are areas of concern that will be investigated during the audit, along with any similar potential issues:

- 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;
- 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

We reviewed the Atomic Swap contracts for many of the common mistakes made in smart contract design and implementation. We found all of the contracts to be well designed and thought out.

The Atomic Swap (Ethereum) and ERC-20 contract code is well organized. In particular, both contracts make good use of truffle testing. We also found that Atomic Swap (Ethereum) and ERC-20 contracts utilize the latest versions of OpenZeppelin, or older versions if reasonable to do so. Both sets of contracts use a declarative style that lends itself well to smart contract design, using current industry best practices for security, including re-entrancy guards, and good validity checks before code execution.

The Tezos Atomic Swap (Tezos) and the FA1.2 contracts are also well organized, and contain Python unit tests with adequate coverage.

Our team found that comments are absent in all contracts and recommend they be included for better readability and comprehension. While maintaining simple code that is self explanatory is desirable to avoid overly complex smart contract code, labels for what the intended usage of functions and the parameters will allow new readers to understand critical functionality more easily (Suggestion 1, Suggestion 3). Furthermore, the FA1.2 contract makes use of a number of functions that appear to no longer be documented which may result in unpredictable behavior or generate cryptic and unhelpful error messages in the event of a failure (Issue D).

While we found the article, Atomex: cross-chain atomic swaps on practice, to be helpful, there is currently no design documentation available for any of the contracts. However, the development team has indicated that documentation creation is in progress. Our team believes that detailed documentation facilitating better understanding of the correctness of the implementation, outlining the API, and the intended and expected behavior of the contracts is prudent and its development and completion should be prioritized (Suggestion 1).

Overall, it is clear the security was strongly considered, particularly since the project supports custom tokens such as ERC-20 and FA1.2 that require careful analysis and threat assessment in comparison to native currency tokens. For example, ERC-20 and FA1.2 tokens require a contract to be formed and called by the swap contract to do transfers, where it may be possible that incorrect addresses are supplied for these tokens, and failed transfers would occur. Furthermore, the usage of re-entrancy guards, checking for contract code presence in the token addresses of the ERC-20 implementation of swaps, safe ERC-20, and modifiers of access control demonstrate a thorough and thoughtful consideration of known security best practices. The addition of documentation, comments and consistency in checks and modifiers will further adhere to best practices and enhance the security of the contracts.

Although our audit scope coverage was extensive and allowed us to investigate and analyze all aspects of the Atomic Swap comprehensively at the contract level, the Atomic Swap Client Core Library was out of scope. We recommended that a review of the core library be completed in order to determine whether the client core is using the contracts in a safe and reliable way.

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: Updating Ownership of Solidity Fiat Token to Mistaken Address	Resolved
Issue B: Atomic Swap (Tezos) and FA1.2 Contracts Have Unclear or Absent Error Messages	Resolved
Issue C: Inconsistencies In Contract Modifier Requirements	Resolved
Issue D: FA1.2 Contract Makes Use of Many Deprecated or Undocumented Functions	Resolved
Suggestion 1: Improved Documentation for Atomic Swap (Ethereum) Contract	Unresolved
Suggestion 2: Consistency in Redeem Times in the Atomic Swap (Ethereum)	Resolved

and ERC-20 Contracts	
Suggestion 3: Comments in the Michelson Code of the Atomic Swap (Tezos) Contract	Resolved

Issue A: Updating Ownership of Solidity Fiat Token to Mistaken Address

Location

https://github.com/atomex-me/atomex-erc20-solidity/blob/fe568c33dfffce6c495faf71af8a11ce6dad00f1/contracts/FiatTokenV1.sol#L53

Synopsis

In the Solidity fiat token implementation, updating the owner of the token may provide a mistaken address that is not controlled by the organization that is intended to control the token. The `transferOwnership()` function checks that the address provided as the new owner is not zero. However, if an incorrect address is provided to this function, then control of the token will be lost.

Impact

High. The impact of this mistake is significant as control of the token will be lost if an address is provided for which the organization does not have a corresponding private key.

Preconditions

An accidental address must be authorized by an Ethereum transaction from the current valid owner of the fiat token.

Feasibility

The likelihood of providing an account that is incorrect during the transfer of ownership is low. Transfer of ownership of a fiat token may not happen frequently or may be controlled by a multisig account where many individuals must agree that the new account is accurate.

Mitigation

A two step method to transfer ownership, while costing a bit of extra gas, could prevent a rare case where ownership is transferred to an account that is not controlled by the intended organization. This process would have two functions that would be called to transfer ownership. The first step is to propose a new owner, modified to have access only by the current owner. This will store the potential new owner in the token contracts state. The second step is to call a function that accepts the new owner that is modified to only have access to the new owner's account. As a result, if the account being updated to is not controlled, the current owner may still reverse the mistake and try again.

Status

The `Ownable.sol` contract has been updated with the suggested mitigation. The contract now has the ability to propose and accept ownership transfers with the functions `proposeOwner()` and `acceptOwnership()`.

Verification

Resolved.

Issue B: Atomic Swap (Tezos) and FA1.2 Contracts Have Unclear or Absent Error Messages

Location

https://github.com/atomex-me/atomex-fa12-ligo/blob/142c29346d010301dcb1f930d46ccd0d98a462b3/src/atomex.ligo

https://github.com/atomex-me/atomex-michelson/blob/82f452d1ea4d0263b7a4eaab782a6e02b06bcaf3/src/atomex.tz

Synopsis

Empty strings instead of a reason for a failed transaction should be provided are present in multiple locations of the FA1.2 contract. For example, the contract will fail with an empty string if a user attempts to redeem before the timeout or tries to redeem with an incorrect secret seed. The FA1.2 contract also makes use of deprecated functions which generate highly cryptic error messages on failure.

Although the Atomic Swap (Tezos) contract has error messages, further clarification about the cause of the error is recommended. For example, if the source of the transaction fails, the check on atomex.tz line 79 results in a message stating “wrong sender address" without further explanation on the issue.

Impact

When a transaction executes in a way that will fail, the client software or initiator of the transaction may have a difficult time identifying the failure case. While this does not create an immediate security problem, it could cause client software or users to be unable to take appropriate actions to correct a mistake.

Remediation

Provide the appropriate corresponding reasoning for failures as opposed to empty strings.

Status

This suggested remediation has been implemented and the Atomic Swap (Tezos) and FA1.2 contracts now have clear error messages for expected failure modes.

Verification

Resolved.

Issue C: Inconsistencies In Contract Modifier Requirements

Location

https://github.com/atomex-me/atomex-solidity/blob/e5d4c03b4bcd735a0cb456cd99ec6d68cfbb98de/contracts/AtomicSwap.sol#L91

https://github.com/atomex-me/atomex-erc20-solidity/blob/fe568c33dfffce6c495faf71af8a11ce6dad00f1/contracts/AtomicSwap.sol#L1500

https://github.com/atomex-me/atomex-fa12-ligo/blob/142c29346d010301dcb1f930d46ccd0d98a462b3/src/atomex.ligo#L40-L47

https://github.com/atomex-me/atomex-michelson/blob/82f452d1ea4d0263b7a4eaab782a6e02b06bcaf3/src/atomex.tz#L29

Synopsis

The modifiers for how a function should restrict the access to the function body are not consistent among the different contract implementations. For example:

- The Atomic Swap (Ethereum) contract does not check that the provided `participant` is non-zero;
- The Atomic ERC-20 contract checks that the `participant` is non-zero;
- The FA1.2 contract checks that only the `participant` is not the source of the transaction; and
- The Atomic Swap (Tezos) contract only checks that the `participant` exists and has proper typing.

Impact

The intended requirements for the participants of a swap's address are not clear. Some contracts have strict requirements for address validation while others do not, which implies that some contracts lack guarantees that a participant address is the intended address. Without stricter requirements, an accidental address could be provided and go unnoticed. This could lead to a case where a redemption is not possible on one end of the swap, while it is possible on the other.

Preconditions

A mistaken participant address must be supplied.

Feasibility

This is only moderately feasible as any given party may witness the mistake and not complete a swap and await a refund.

Remediation

Ensure that all participant address requirements are satisfied in each implementation of the swaps contracts.

For the Atomic Swap (Ethereum) and ERC-20 contracts, ensure that there is a check for a non-zero address provided.

Status

Following a refactor by the Atomex team, the Atomic Swap (Ethereum) contract has been updated to include a check to ensure that non-zero addresses are present in the participant parameter, making this consistent with the edge case checking done in the ERC-20 contracts.

After further discussion with the Atomex team in which they state that the only undesirable address in the parameter is the sender, we agree the Atomic Swap (Tezos) and FA1.2 contract checks are adequate and do not need to be consistent or present in the initialization functions of the Atomic Swap (Tezos) contract.

Verification

Resolved.

Issue D: FA1.2 Contract Makes Use of Many Deprecated or Undocumented Functions

Location

https://github.com/atomex-me/atomex-fa12-ligo/blob/142c29346d010301dcb1f930d46ccd0d98a462b3/src/atomex.ligo

Synopsis

The FA1.2 contract makes use of a number of functions that are deprecated in official PascaLIGO documentation, or are entirely absent from the latest version of it.

Impact

Deprecated and undocumented functions, especially when used to avoid error handling, can behave in ways that are hard to predict or may change over time. In cases of failure, the above functions will often generate cryptic and unhelpful error messages. For the functions with no documentation, it is impossible to have confidence in the behavior of the function, especially for edge cases or in future updates of the PascaLIGO compiler.

Technical Details

In many cases, these are functions that have the potential to fail, but avoid handling the failure. For example, `get_entrypoint` will fail if pointed to a contract without the specified entrypoint with a confusing error message such as "bad address for get_entrypoint (%transfer)". The PascaLIGO reference suggests `Tezos.get_entrypoint_opt`, which requires explicit handling of the failure case. In the case of the function `get_force`, it seems to be entirely absent from the PascaLIGO documentation.

Mitigation

In general, we noticed the following deprecated functions, and provide suggested replacements:

Deprecated Function	Replacement
`get_entrypoint`	`Tezos.get_entrypoint_opt`
`transaction`	`Tezos.transaction`
`size`	`Bytes.length`
`source`	`Tezos.source`
`get_force`	`Big_map.find_opt`

Status

This issue has been resolved by removing references to deprecated functions.

Verification

Resolved.

Suggestions

Suggestion 1: Improved Documentation for Atomic Swap Contract APIs

Location

All contracts.

Synopsis

There is limited documentation for the Atomic Swap (Ethereum), ERC-20, Atomic Swap (Tezos), and FA1.2 contracts. While there are links provided to the OpenZeppelin repositories for the source of the Ethereum token code, there are no comments provided for any of the code's functionality or intended usage. A blog post provides an overview of expected functionality, but the expected operation of the actual contract entrypoints is ambiguous.

For example, we initially found it difficult to understand the countdown and payoff logic without documentation. The countdown state is only used for the ERC-20 contract and not the Atomic Swaps (Ethereum) contract. Additional explanations would help facilitate better understanding and appreciation of this feature.

Remediation

Atomic Swap (Ethereum) and ERC-20 contracts: adhere to the standard Solidity comment format and provide comments for each function in the form of its inputs and intended usage.

Atomic Swap (Tezos) and FA1.2 contracts: create a design document with the expected API and possible failure modes for all contracts.

Status

The Atomex team has stated that the creation of documentation is in progress and that they intend to develop both technical articles in addition to introductory content, as recommended.

Verification

Unresolved.

Suggestion 2: Consistency in Redeem Times in the Atomic Swap (Ethereum) and ERC-20 Contracts

Location

Atomic Swap (Ethereum) and ERC-20 contracts.

Synopsis

In the Atomic Swap (Ethereum) and ERC-20 contracts, the time at which a redeem is considered valid is somewhat inconsistent. The Atomic Swap (Ethereum) enforces that the blocktime is strictly less than the refund time:

```
require(block.timestamp < swaps[_hashedSecret].refundTimestamp
```

The ERC-20 version allows for the block timestamp to be equal to the refund timeout:

```
require(block.timestamp <= swaps[_hashedSecret].refundTimestamp
```

This should have no impact on security.

Mitigation

Make both either equal to or less than or equal to block timestamps.

Status

Time validations have been updated to use a uniform operator check for the Atomic Swap (Ethereum) and ERC-20 contracts.

Verification

Resolved.

Suggestion 3: Comments in Michelson Code of the Atomic Swap (Tezos) Contract

Location

https://github.com/atomex-me/atomex-michelson/blob/82f452d1ea4d0263b7a4eaab782a6e02b06bcaf3/src/atomex.tz

Synopsis

The Michelson code can be unclear, especially when looking at a single branch of a conditional statement. Without comments, it can be difficult to tell what any given code block in Michelson is intended to be doing, without reading and understanding the entire contract. This can result in requiring a significant amount of time to make small code changes, especially for someone unfamiliar with the contract code.

Mitigation

A small number of comments to indicate which code block corresponds to which entrypoint would be helpful. For longer code blocks, comments indicating what procedure is being accomplished over the next few lines would also be valuable.

Status

This issue has been resolved by adding comments showing the structure of the Michelson program to the Atomic Swap (Tezos) contract, as recommended.

Verification

Resolved.

Recommendations

We recommend that the remaining *Suggestion* stated above be addressed as soon as possible and that further audits are considered if future changes are made to the contracts.

We commend the following of security best practices and encourage continued efforts as suggested in the *General Comments*.

Additionally, the Atomic Swap Client Core Library should be reviewed as it was out of scope for this audit.

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.