apriorit

TEZEX

Security audit report

Prepared for Cryptonomic
April 26, 2021

Table of contents

Project summary

Name	TEZEX
Source	<table><tr><th>Repository</th><th>Revision</th></tr><tr><td>https://github.com/StableTechnologies/TEZEX</td><td>branch/master - 838f664</td></tr></table>
Methods	Code review, Static analysis, Manual penetration testing

Coverage and scope of work

The audit was focused on an in-depth analysis of the token swap contract implementation, as well as the bot application that interacts with those contracts, and the front-end core library.

We have conducted smart-contract audit under the following criteria:

- Behavioural analysis of smart contract source code
- Penetration testing
- Unit test coverage analysis
- Manual code review and evaluation of code quality
- Analysis in regards to host network

The bot application audit includes:

- Manual code review
- Static analysis
- Analysis of interaction with other resources
- Filesystem and local resources usage analysis
- Analysis of configurations and dependencies

The front-end core library audit was conducted under the following criteria:

- Manual code review
- Static analysis
- Analysis of interaction with blockchain
- Private user data handling analysis

The audit was performed using manual code review and automated static analysis. Once some potential vulnerabilities were discovered, manual attacks were performed to check if they can be easily exploited.

Smart contracts overview

Apriorit conducted a security assessment of TEZEX token swap smart-contracts to evaluate its current state and risk posture.

This security assessment was conducted in April 2021 to evaluate the exposure to known security vulnerabilities, to determine potential attack vectors and to check if any of them can be exploited maliciously.

Contracts description and functionality

The audited contracts are implementing the logic of atomic cross-chain token swap. The following files were audited:

File	Description
TokenSwap.sol	Ethereum implementation of swap logic
tokenSwap.js	Tests for Ethereum smart contract
TokenSwap.py	Tezos smart contract and tests

The behavior of the contract's functions was analyzed in comparison to the description of public documentation (https://arxiv.org/pdf/1801.09515.pdf and http://www.cse.unsw.edu.au/~meyden/research/swap.pdf). No inconsistencies regarding the cross-chain token swap scenario were found. There is an additional logic in Tezos smart contract that allows to set a reward parameter.

The following functions are required for both contracts to create an atomic swap:

- The ability to initiate new swap, providing the required amount of tokens, hash of the secret and refund timestamp
- The ability to add counterparty
- The ability to redeem token by providing raw secret data

- The ability to refund tokens if redeem time is expired.

Access restriction:

Function	**Restriction**
Add counterparty	Only by initiator
Redeem	No access restrictions. Tokens will be sent to provided counterparty
Refund	No access restrictions. Tokens will be sent to initiator

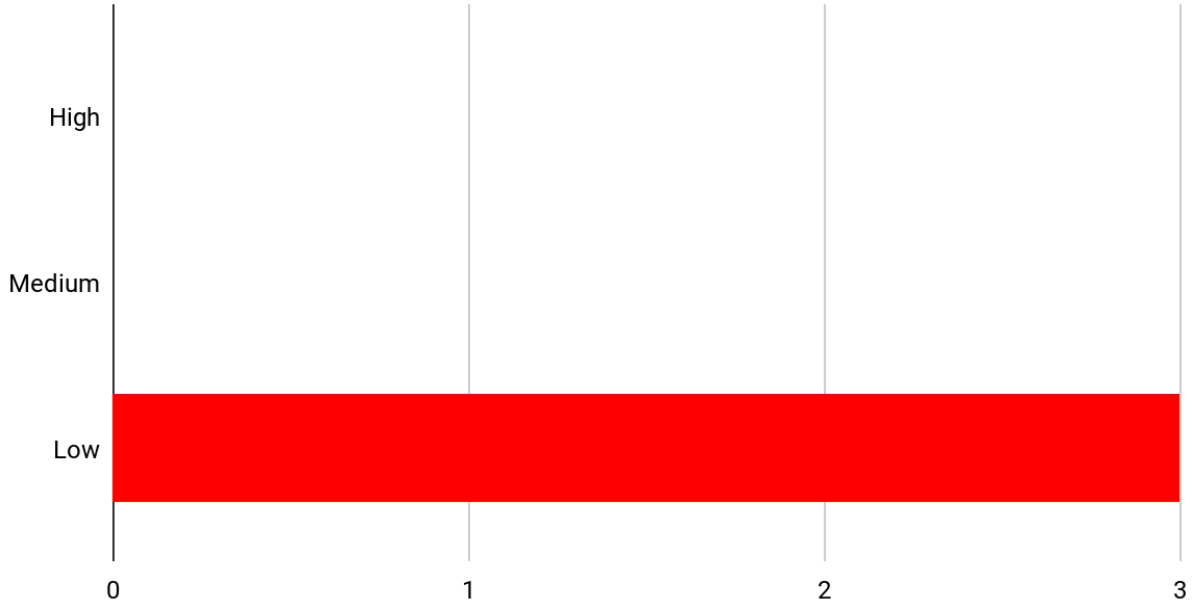
Summary of discovered vulnerabilities

During the assessment no high risk vulnerabilities were discovered which indicated good attention to security in the implementation.

Overall, only 3 low risk vulnerabilities were discovered. Such statistics are acquired due to the audited contract's simplicity - almost all logic implemented in one contract, communication with other contracts (ERC20, FA1.2) is performed securely.

For more detailed information on all of the findings discovered, refer to the detailed findings section (Appendix A: Detailed Findings) of the report.

Vulnerability chart



Risk Rating	Finding Name	Recommendation	Status
Low risk	Prefer external to public visibility level	Changing visibility level to external increases code readability. Moreover, in many cases functions with an external visibility modifier spend less gas compared to functions with a public visibility modifier.	OPEN
Low risk	Implicit visibility level	The default function visibility level in contracts is public, in interfaces - external, state variable default visibility level is internal. In contracts, the fallback function can be external or public. In interfaces, all the functions should be	OPEN

		declared as external. The function visibility should be explicitly defined to prevent confusion.	
Low risk	Integer Overflow	Usage of a safemath library should be considered for arithmetic operations to prevent the possibility of integer overflow.	OPEN

Unit tests coverage

The main execution path and some of the alternative scenarios are covered by unit tests. It is recommended to cover all scenarios to prevent logic error if the code would be updated in the future.

Uncovered scenarios for the Ethereum smart contract:

- Check that refund is not available after redeem.
- Check that redeem is not available after refund.
- Check that the refund is available for the swap in wait state.
- Check that counterparty cannot be added for the swap in the initiated state.

Uncovered scenarios for the Tezos smart contract:

- Check that refund is not available after redeem.
- Check that redeem is not available after refund.
- Check that the refund is available for the swap in wait state.
- Check that counterparty cannot be added for the swap in the initiated state.
- Check that swap can not be initiated if there is not enough amount of approved tokens.

Security rating

Apriorit reviewed Cryptonomic security posture in regards to the TEZEX smart contracts, and Apriorit consultants identified security practices that are strengths as well as vulnerabilities that create low risks. Taken together, the combination of asset criticality, threat likelihood and vulnerability severity have been combined to assign an assessment grade for the overall security of the application. An explanation of the grading scale is included in the second table below.

In conclusion, Apriorit recommends that Cryptonomic continues to follow good security practices that are already established and further improves security posture by addressing all of the described findings.

	High	**Medium**	**Low**	**Security**	**Grade**
TEZEX smart contracts	0	0	3	Highly Secure	A

Security grading criteria

Grade	Security	Criteria Description
A	Highly Secure	Exceptional attention to security. No high or medium risk vulnerabilities with a few minor low risk vulnerabilities.
B	Moderately Secure	Good attention to security. No high risk vulnerabilities with only a few medium or several low risk vulnerabilities.
C	Marginally Secure	Some attention to security, but security requires improvement. A few high risk vulnerabilities were identified and can be exploited.
D	Insecure	Significant gaps in security exist. A large number of high risk vulnerabilities were identified during the assessment.

Bot application overview

The bot application is a part of TEZEX project that performs monitoring of active swaps, responds to suitable swaps and activates refund for outdated swaps. The application is written in javascript.

In April 2021, Apriorit conducted a security assessment of the bot application to evaluate the exposure to known security vulnerabilities, to determine potential attack vectors and to check if any of them can be exploited maliciously.

Summary of discovered vulnerabilities

The analysis of the code shows that implementation was focused on providing a secure solution - vulnerable data is encrypted and interaction with external resources is performed through secure connection. Overall analysis showed no high risk vulnerabilities.

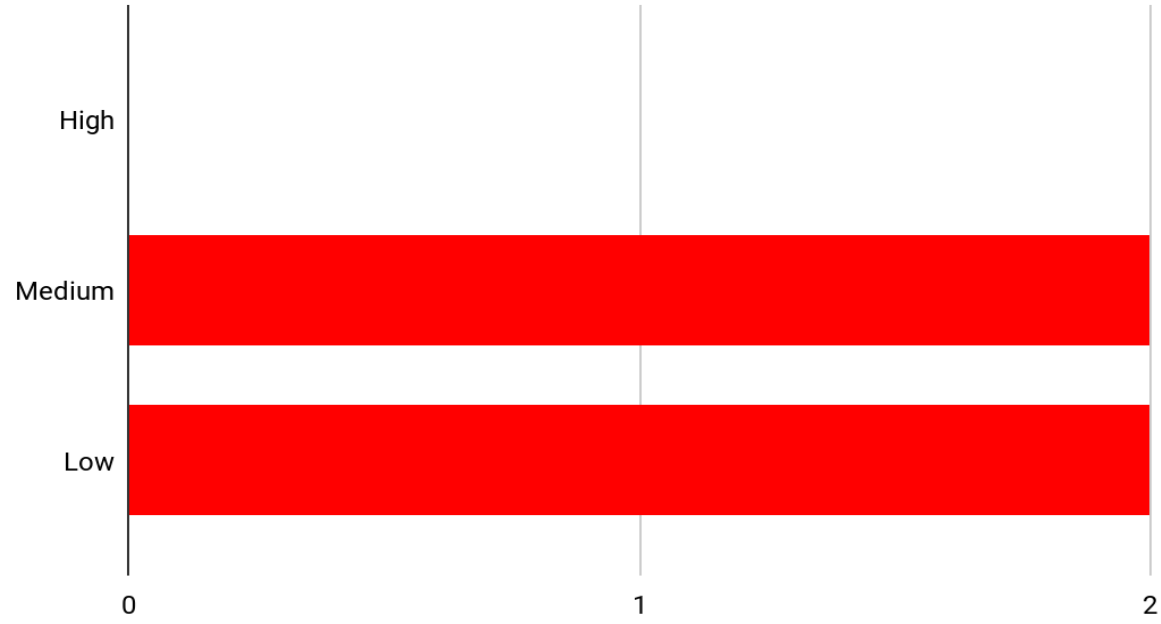
Static analysis showed the presence of multiple problems regarding code style and type analysis. Apriorit recommends to use linters during development to prevent such errors.

Manual code review showed logical error, where the reward value from usdc/usdtz smart contract was used for all swap pairs.

The refund logic may cause high costs, as every refund spends coins to pay transaction fees. This is not really an exploit, because the attacker would spend nearly as much as the bot, but with a high usage, bot can cause some losses.

Automatic dependency analysis showed one low vulnerability and another medium vulnerability was found during manual analysis. For more detailed information on all of the findings discovered, refer to the detailed findings section (Appendix A: Detailed Findings) of the report.

Vulnerability chart



Risk Rating	Finding Name	Recommendation	Status
Medium risk	Logical error - usage of reward value	Reward value should be received for each swap pair.	OPEN
Medium risk	External dependency with vulnerability (node-fetch)	It is recommended to update node-fetch dependency to v2.6.1 as it is marked as an important security release. Currently, v2.6.0 is in use.	OPEN

Low risk	External dependency with vulnerability (web3)	All versions of web3 are vulnerable to Insecure Credential Storage. No fix is currently available. Consider using an alternative module until a fix is made available.	OPEN
Low risk	Type-safe operations and code style	Usage of linters should be considered to preview and fix such problems. Some of the issues can be fixed automatically while others need manual modifications.	OPEN

Security rating

Apriorit reviewed Cryptonomic security posture in regards to the TEZEX bot application, and Apriorit consultants identified security practices that are strengths as well as vulnerabilities that create medium and low risks. An explanation of the grading scale is included in the second table below.

In conclusion, Apriorit recommends the usage of linters during development of javascript applications.

	High	**Medium**	**Low**	**Security**	**Grade**
TEZEX bot application	0	2	2	Moderately Secure	B

Security grading criteria

Grade	Security	Criteria Description
A	Highly Secure	Exceptional attention to security. No high or medium risk vulnerabilities with a few minor low risk vulnerabilities.
B	Moderately Secure	Good attention to security. No high risk vulnerabilities with only a few medium or several low risk vulnerabilities.
C	Marginally Secure	Some attention to security, but security requires improvement. A few high risk vulnerabilities were identified and can be exploited.
D	Insecure	Significant gaps in security exist. A large number of high risk vulnerabilities were identified during the assessment.

Front-end core library overview

Apriorit conducted a security assessment of the front-end core library of TEZEX project. This library provides functionality to create a new swap.

This security assessment was conducted in April 2021 to evaluate the exposure to known security vulnerabilities, to determine potential attack vectors and to check if any of them can be exploited maliciously.

Summary of strengths

Manual code review and testing showed that the core library is well maintained and the code standards are noticeably high. All functions are well documented with JSDoc markup language.

The interaction with blockchain is performed in a secure way. The usage of user data in the core library is limited strictly for communication with blockchain and there is no data leakage.

Core library is a part of the front-end project with shared dependencies. Automatic dependency analysis revealed 4307 vulnerabilities (3595 low, 362 moderate, 350 high). These vulnerabilities are not specifically related to the core library but to the whole front-end project. The explicit list of them can be received with the "npm audit" command.

Security rating

Apriorit reviewed Cryptonomic security posture in regards to the TEZEX front-end core library. An explanation of the grading scale is included in the second table below.

In conclusion, Apriorit recommends that Cryptonomic continues to follow good security practices that are already established.

	High	Medium	Low	Security	Grade
TEZEX front-end core library	0	0	0	Highly Secure	A

Security grading criteria

Grade	Security	Criteria Description
A	Highly Secure	Exceptional attention to security. No high or medium risk vulnerabilities with a few minor low risk vulnerabilities.
B	Moderately Secure	Good attention to security. No high risk vulnerabilities with only a few medium or several low risk vulnerabilities.
C	Marginally Secure	Some attention to security, but security requires improvement. A few high risk vulnerabilities were identified and can be exploited.
D	Insecure	Significant gaps in security exist. A large number of high risk vulnerabilities were identified during the assessment.

Appendixes

Appendix A. Detailed findings

Risk rating

Our risk ratings are based on the same principles as the Common Vulnerability Scoring System. The rating takes into account two parameters: exploitability and impact. Each of these parameters can be rated as high, medium or low.

Exploitability - What knowledge the attacker needs to exploit the system and what pre-conditions are necessary for the exploit to work:

- **High** - Tools for the exploit are readily available and the exploit requires no specialized knowledge about the system.
- **Medium** - Tools for the exploit available but have to be modified. The exploit requires some specialized knowledge about the system.
- **Low** - Custom tools must be created for the exploit. In-depth knowledge of the system is required to successfully perform the exploit.

Impact - What effect the vulnerability will have on the system if exploited:

- **High** - Administrator level access and arbitrary code execution or disclosure of sensitive information (private keys, personal information).
- **Medium** - User level access with no disclosure of sensitive information.
- **Low** - No disclosure of sensitive information. Failure to follow recommended best practices that does not result in an immediately visible exploit.

Based on the combination of the parameters the overall risk rating is assigned to the vulnerability.

Smart contracts discovered vulnerabilities

Low risk - Public visibility level of the functions

Description:

In the code, there are functions with the public visibility level that are not called internally.

Changing visibility level to external increases code readability. Moreover, in many cases functions with an external visibility modifier spend less gas compared to functions with a public visibility modifier.

Affected code:

The following functions in Ethereum smart contract:

- addCounterParty
- redeem
- initiateWait
- refund
- toggleContractState
- getAllSwaps

Recommendation:

Visibility level of listed functions should be changed from public to external.

Low risk - Implicit visibility level

Description:

The default function visibility level in contracts is public, in interfaces - external, state variable default visibility level is internal. In contracts, the fallback function

can be external or public. In interfaces, all the functions should be declared as external.

Affected code:

The “address payable admin” variable in Ethereum smart contract.

Recommendation:

Function and state variables visibility should be explicitly defined to prevent confusion.

Low risk - Integer Overflow

Description:

Integer overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented – either larger than the maximum (overflow) or lower than the minimum value (underflow). In Solidity once overflow occurs, the value loops around. In case of overflow it goes from large values to zero and in case of underflow - from 0 to extremely large values. This is especially dangerous when the value in question is the amount of user's tokens.

Affected code:

The usage of “uint256 public count” variable in Ethereum smart contract.

Recommendation:

Usage of unchecked arithmetics should be avoided, even if some preconditions for values are checked. It is recommended to use a dedicated library for safe arithmetics.

Bot application discovered vulnerabilities

Medium risk - Logical error - usage of reward value

Description:

The reward value is received only once from usdc/usdtz swap contract and applied to all other swap pairs.

Affected code:

Reward value is received in getBotFees() function.

Recommendation:

Reward value should be received for each swap pair.

Medium risk - External dependency with vulnerability (node-fetch)

Description:

The bot application has dependency to node-fetch v2.6.0 that is vulnerable to Denial of Service.

Recommendation:

The version of node-fetch should be upgraded to v2.6.1

Low risk - External dependency with vulnerability (web3)

Description:

All versions of web3 are vulnerable to Insecure Credential Storage. The package stores encrypted wallets in local storage and requires a password to load the

wallet. Once the wallet is loaded, the private key is accessible via LocalStorage. Exploiting this vulnerability likely requires a Cross-Site Scripting vulnerability to access the private key.

Recommendation:

No fix is currently available. Consider using an alternative module until a fix is made available.

Low risk - Type-safe operations and code style

Description:

The bot application has multiple problems that can be found and fixed by the usage of linters. Those problems includes, but not limited to:

- Reference errors in regards of undeclared variables
- Usage of type-safe equality operators
- Import order

Recommendation:

Usage of linters should be considered to preview and fix such problems.

Appendix B. Methodologies description

Smart contracts security checks

Contract vulnerabilities are often introduced due to the semantic gap between the assumptions that contract developers make about the underlying execution semantics and the actual semantics of smart contracts.

Our security checklist for smart contract includes:

- Integer Overflow
- Reentrancy
- Race Conditions
- Unchecked External Call
- Unprotected Function
- Short Address Attack
- Multiple sends in a single transaction
- Delegatecall or callcode to untrusted contract
- Timestamp dependence
- DoS with (Unexpected) revert
- Storage Allocation Exploits
- DoS with Block Gas Limit
- Custom ABI-encoded arrays as input
- Underflow Storage Manipulation
- Combinations of vulnerabilities
- GAS usage analysis

Javascript applications security checks

Our security checklist for javascript applications includes:

- Buffer overflow
- Denial of service
- SQL and NoSQL injection
- Shell injection
- Insecure dependencies
- Input validation
- Cross Site Scripting
- Access control issues
- Unsafe usage of cryptography
- Error handling