USTX

Smart Contract Audit
Final Report

September 27, 2021

Introduction

1. About USTX

The blockchain technology has proven a key development factor for the new economy, allowing for safe, fast and low-cost transfer of value between individuals and companies. The USTX project aim is to deploy a new utility token, based on smart contract technology, that has the growth potential of digital currencies like BTC and the stability effect during bear market conditions typical of stablecoins. This can be obtained by creating a bespoke AMM DEX, that dynamically manages the reserve to allow for price increase in uptrend market while damping the price decrease during downtrend. The reserve is completely transparent and certified by the underlying blockchain, removing the need for complex proof of reserve methodologies. A target reserve level is always enforced by the smart contract to ensure the long term stability of the system. The reserve liquidity is forever locked in the contract, meaning that the owner of smart contract is not allowed to withdraw liquidity. The token and DEX contracts are open source and deployed using the TRC20 standard over the TRON blockchain.

Visit https://ustx.io/ to know more.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 15+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The USTX team has provided the following doc for the purpose of audit:

1. https://ustx.io/wp-content/uploads/2021/09/USTX_WhitepaperV2.pdf

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: USTX
- Contracts Names: UpStableToken, UstxDEX, UstxProxy
- Languages: Solidity(Smart contract)
- Github commit hash for audit: 03b8f9c494a0518fd3a236f1c03af7ae49c64f9e
- Github commit hash for audit: d89949854a3980cd1dd966bdc04790b23540407d
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck.
- Mainnet Contract Address:
    - https://tronscan.io/#/contract/TMtyD8z93TLdMh4Swht8WyMx6R7LzqaH1y/code
    - https://tronscan.io/#/contract/TT7EHYyThW1G1WBndya7RQpNSXFHyzeHfx/code
    - https://tronscan.io/#/contract/TYX2iy3i3793YgKU5vqKxDnLpiBMSa5EdV/code

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

Every issue in this report was assigned a severity level from the following:

Admin/Owner Privileges can be misused either intentionally or unintentionally.

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	-	-	-
Closed	-	-	2

Admin/Owner Privileges

The **admin/owner** of **Ustx** smart contracts has various privileges over the smart contracts. These privileges can be misused either intentionally or unintentionally (in case the admin's private key gets hacked). We assume that these extra rights will always be used appropriately. Some of these admin rights are listed below.

1. **In the UpStableToken contract, the admin address can mint any amount of USTX tokens.**
   The **UpStableToken** contract contains a **mint**() function by which the admin accounts can mint any number of USTX tokens to any address.

   Recommendation:
   Consider hardcoding predefined ranges or validations for input variables in privileged access functions. Also, consider adding some governance for admin rights for smart contracts or use a multi-sig wallet as admin/owner addresses.

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

1. **Unnecessary initialization to zero value.**
   In the **constructor**() of the **AdminRole** contract in **AdminRole.sol**, the **_numAdmins** state variable is initialized to 0.

```
constructor (uint256 minAdmins) internal {
    _numAdmins=0;
    ...
}
```

   In Solidity the default value of **uint256** datatype is 0. So there is no need to explicitly initialize the **_numAdmins** variable to zero.

   Recommendation:
   Consider removing the above-shown statement from the **constructor**().

   Amended (September 27th, 2021): The issue was fixed by the **USTX** team and is no longer present.

2. **Contract code size exceeds 24576 bytes.**
   The **UstxDEX** smart contract's code size exceeds 24576 bytes (24 KB). This contract size limit was introduced in Spurious Dragon. Contracts exceeding this limit become non-deployable on testnet/mainnet without compiler optimization being enabled.

   Recommendation:
   Consider further dividing the contract into sub-contracts/libraries to reduce the code size.

   Acknowledged (September 27th, 2021)

Unit Test

No unit tests were provided by the USTX team.

Recommendation:
Our team suggests that the developers should write extensive test cases for the contracts.

Coverage Report

Coverage report cannot be generated without unit test cases.

Recommendation:
We recommend 100% line and branch coverage for unit test cases.

Automated Auditing

Solhint Linting Violations

Solhint is an open-source project for linting solidity code, providing both security and style guide validations. It integrates seamlessly into most mainstream IDEs. We used Solhint as a plugin within our VScode for this analysis. Multiple Linting violations were detected by Solhint, it is recommended to use Solhint's npm package to lint the contracts.

Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Solidity smart contracts, with security analysis applied to them in real-time. We performed analysis using Contract Library on the testnet address of the UpStableToken, UstxDEX and UstxProxy contracts used during manual testing:

- UpStableToken: 0x8cE237CAc764F91D9f51F7D6383C77aE545CfAFA
- UstxDEX: 0x0c9d556057b23f35B5805ec93999283eD14a379D
- UstxProxy: 0xc4Cf14Db464EEf91e4EA398Bc06Ab5a1A5ea8ffF

It raises no major concern for the contracts.

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Solidity smart contracts and has critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

It raises no major concern for the contracts.

```
INFO:Detectors:
ERC20Detailed.constructor(string,string,uint8).name (ERC20Detailed.sol#21) shadows:
        - ERC20Detailed.name() (ERC20Detailed.sol#30-32) (function)
ERC20Detailed.constructor(string,string,uint8).symbol (ERC20Detailed.sol#21) shadows:
        - ERC20Detailed.symbol() (ERC20Detailed.sol#38-40) (function)
ERC20Detailed.constructor(string,string,uint8).decimals (ERC20Detailed.sol#21) shadows:
        - ERC20Detailed.decimals() (ERC20Detailed.sol#54-56) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Pragma version^0.5.0 (AdminRole.sol#4) allows old versions
Pragma version^0.5.0 (Context.sol#4) allows old versions
Pragma version^0.5.0 (ERC20.sol#4) allows old versions
Pragma version^0.5.0 (ERC20Detailed.sol#4) allows old versions
Pragma version^0.5.0 (IERC20.sol#4) allows old versions
Pragma version^0.5.0 (Pausable.sol#4) allows old versions
Pragma version^0.5.0 (Roles.sol#4) allows old versions
Pragma version^0.5.0 (SafeMath.sol#4) allows old versions
Pragma version^0.5.0 (UpStableToken.sol#3) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Redundant expression "this (Context.sol#27)" inContext (Context.sol#16-30)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
addAdmin(address) should be declared external:
        - AdminRole.addAdmin(address) (AdminRole.sol#34-36)
renounceAdmin() should be declared external:
        - AdminRole.renounceAdmin() (AdminRole.sol#43-46)
totalSupply() should be declared external:
        - ERC20.totalSupply() (ERC20.sol#46-48)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (ERC20.sol#53-55)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (ERC20.sol#73-75)
name() should be declared external:
        - ERC20Detailed.name() (ERC20Detailed.sol#30-32)
symbol() should be declared external:
        - ERC20Detailed.symbol() (ERC20Detailed.sol#38-40)
decimals() should be declared external:
        - ERC20Detailed.decimals() (ERC20Detailed.sol#54-56)
paused() should be declared external:
        - Pausable.paused() (Pausable.sol#42-44)
pause() should be declared external:
        - Pausable.pause() (Pausable.sol#65-68)
unpause() should be declared external:
        - Pausable.unpause() (Pausable.sol#73-76)
mint(address,uint256) should be declared external:
        - UpStableToken.mint(address,uint256) (UpStableToken.sol#52-55)
burn(uint256) should be declared external:
        - UpStableToken.burn(uint256) (UpStableToken.sol#62-64)
burnFrom(address,uint256) should be declared external:
        - UpStableToken.burnFrom(address,uint256) (UpStableToken.sol#71-73)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Concluding Remarks

While conducting the audits of the USTX smart contracts, it was observed that the contracts contain only Low severity issues.

Our auditors suggest that Low severity issues should be resolved by USTX developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the USTX platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes