# Tribex

## Smart Contract Security Audit

Prepared by ShellBoxes

March 31st, 2022 - May 5th, 2022

Shellboxes.com

contact@shellboxes.com

## Document Properties

| | |
|---|---|
| Client | Tribex |
| Version | 1.0 |
| Classification | Public |

## Scope

The Tribex  Contract deployed in the Ethereum Mainnet

| Contract Name | Contract Address |
|---|---|
| txtest.sol | 0xff9981d2c6c6d612e03e4a32f5488e552eeae285 |

## Contacts

| COMPANY | EMAIL |
|---|---|
| ShellBoxes | contact@shellboxes.com |

# Contents

# 1 Introduction

Tribex engaged ShellBoxes to conduct a security assessment on the Tribex beginning on March 31st, 2022 and ending May 5th, 2022. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

## 1.1 About Tribex

Tribex is A metaverse brand built by the community and for the community.
Tribe X Empire is an ambitious NFT collectible featuring 11,111 exclusive avatars that will live on the Ethereum Blockchain. their aim is to bridge the tech gap and empower underrepresented communities. By holding a Tribe X Empire NFT, users will be eligible for giveaways, airdrops, whitelisted for future drops, access to members-only benefits/experiences, access to the Tribe X Ecosystem and more.

| | |
|---|---|
| Issuer | Tribex |
| Website | `https://www.iamtribex.com` |
| Type | Solidity Smart Contract |
| Audit Method | Whitebox |

## 1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

### 1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

| Impact | | | | |
|---|---|---|---|---|
| | High | Critical | High | Medium |
| | Medium | High | Medium | Low |
| | Low | Medium | Low | Low |
| | | High | Medium | Low |

Likelihood

# 2 Findings Overview

## 2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Tribex implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

## 2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 1 high-severity, 1 medium-severity, 3 low-severity vulnerabilities.

| Vulnerabilities | Severity | Status |
|---|---|---|
| A.1. User Can Buy With A Higher Price | HIGH | Acknowledged |
| A.2. Missing Value Verification | MEDIUM | Acknowledged |
| A.3. Race Condition | LOW | Acknowledged |
| A.4. Renounce Ownership | LOW | Acknowledged |
| A.5. Floating Pragma | LOW | Acknowledged |

# 3 Finding Details

## A txtest.sol

### A.1 User Can Buy With A Higher Price [HIGH]

#### Description:

In the mintOGSale and mintWLSale functions, the users that are included in the presale can mint a quantity of tokens after paying enough native tokens. If the user pays more than numberOfMints * price there is no way to get this amount back.

#### Code:

Listing 1: txtest.sol

```
98  require(
99      msg.value >= numberOfMints * OGprice,
100     "Not enough ether to mint, please add more eth to your wallet"
101 );
```

Listing 2: txtest.sol

```
139 require(
140     msg.value >= numberOfMints * WLprice,
141     "Amount of ether is not enough to mint, please send more eth based
        ↪ on price"
142 );
```

Listing 3: txtest.sol

```
166 require(
167     msg.value >= numberOfMints * price,
168     "Amount of ether is not enough, please add more eth"
169 );
```

### Risk Level:

Likelihood – 4
Impact - 4

### Recommendation:

It is recommended to make sure the user pays only the required amount for the mint.

### Status - Acknowledged

The Tribex team has acknowledged the risk stating that the input is controlled by the front-end user interface (mint.iamtribex.com)and the users will only pay the required amount for the mint. However, the front-end checks are not enough to remediate the risk as it can be easily bypassed.

## A.2 Missing Value Verification [MEDIUM]

### Description:

Certain functions lack a safety check in the values, the values of the arguments should be verified to allow only the ones that go with the contract's logic. The maxSupply variable should only be allowed to get updated to a higher value, if not, it will cause a conflict and it will be possible to have the number of minted tokens be higher than the max supply. In addition to that, the value of the prices should be verified to be different than zero.

### Code:

Listing 4: txtest.sol

```
300 //change the supply limit
301 function changeSupplyLimit(uint256 _new) external onlyOwner {
302     maxSupply = _new;
303 }
304 //set public mint price
305 function setOGprice(uint256 _new) external onlyOwner {
```

```
306     OGprice = _new;
307 }
308 function setWLprice(uint256 _new) external onlyOwner {
309     WLprice = _new;
310 }
311 function setMintPrice(uint256 _new) external onlyOwner {
312     price = _new;
313 }
```

### Risk Level:

Likelihood – 2
Impact - 4

### Recommendation:

It's recommended to verify the values provided in the arguments. The concerns can be resolved by utilizing a require statement.

### Status - Acknowledged

The Tribex team has acknowledged the risk stating that the input is controlled by the front-end user interface (mint.iamtribex.com) and users are limited and only able to mint per maxSupply, but even with the front-end check , it is still not enough to eliminate the risk.

## A.3 Race Condition [LOW]

### Description:

The prices of mints are modifiable by the owner, if a user mint a quantity of tokens then the owner changes the price, there will be a possibility that the owner's transaction gets mined first, therefore the user's transaction will get executed using the new price which will cause the user to pay an unexpected amount.

## Code:

Listing 5: txtest.sol

```
98  require(
99      msg.value >= numberOfMints * OGprice,
100     "Not enough ether to mint, please add more eth to your wallet"
101 );
```

Listing 6: txtest.sol

```
139 require(
140     msg.value >= numberOfMints * WLprice,
141     "Amount of ether is not enough to mint, please send more eth based on
        ↪  price"
142 );
```

Listing 7: txtest.sol

```
166 require(
167      msg.value >= numberOfMints * price,
168      "Amount of ether is not enough, please add more eth"
169 );
```

## Risk Level:

Likelihood – 2
Impact - 4

## Recommendation:

It's recommended to verify the values provided in the arguments. The concerns can be resolved by utilizing a require statement.

## Status - Acknowledged

The Tribex team has acknowledged the risk.

## A.4 Renounce Ownership [LOW]

### Description:

Typically, the contract's owner is the account that deploys the contract. As a result, the owner can perform certain privileged activities. The renounceOwnership function is used in smart contracts to renounce ownership. However, if the contract's ownership has never been transferred before renouncing it, it will never have an Owner, which may result in a denial of service.

### Code:

Listing 8: txtest.sol

```
16 contract Empire is ERC721A, Ownable, ReentrancyGuard {
```

### Risk Level:

Likelihood – 1
Impact - 3

### Recommendation:

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method will require two or more users to sign the transaction. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

### Status - Acknowledged

The Tribex team has acknowledged the risk.

## A.5 Floating Pragma [LOW]

### Description:

The contract makes use of the floating-point pragma 0.8.4. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

### Code:

Listing 9: txtest.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.4;
```

### Risk Level:

Likelihood – 2
Impact - 2

### Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

### Status - Acknowledged

The Tribex team has acknowledged the risk.

# 4 Best Practices

## BP.1 Unnecessary variable initialization

### Description:

When a variable is declared in solidity, it gets initialized with its type's default value. Thus, there is no need to initialize a variable with the default value.

### Code:

Listing 10: txtest.sol

```
25 bool public OGsaleActive = false;
26 bool public WLsaleActive = false;
27 bool public saleActive = false;
```

# 5 Static Analysis (Slither)

## Description:

ShellBoxes expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

## Results:

```
Reentrancy in ERC721A._mint(address,uint256,bytes,bool) (contracts/
   ↪ txtest.sol#365-407):
 External calls:
 - ! _checkContractOnERC721Received(address(0),to,updatedIndex ++,_data)
    ↪  (contracts/txtest.sol#393)
  - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data
     ↪ ) (contracts/txtest.sol#570-580)
 State variables written after the call(s):
 - _currentIndex = updatedIndex (contracts/txtest.sol#404)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #reentrancy-vulnerabilities-1

ERC721A._checkContractOnERC721Received(address,address,uint256,bytes) (
   ↪ contracts/txtest.sol#564-581) ignores return value by
   ↪ IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,
   ↪ _data) (contracts/txtest.sol#570-580)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #unused-return

Variable 'ERC721A._checkContractOnERC721Received(address,address,uint256
   ↪ ,bytes).retval (contracts/txtest.sol#570)' in ERC721A.
```

```
    ↪ _checkContractOnERC721Received(address,address,uint256,bytes) (
    ↪ contracts/txtest.sol#564-581) potentially used before declaration
    ↪ : retval == IERC721Receiver(to).onERC721Received.selector (
    ↪ contracts/txtest.sol#571)
Variable 'ERC721A._checkContractOnERC721Received(address,address,uint256
    ↪ ,bytes).reason (contracts/txtest.sol#572)' in ERC721A.
    ↪ _checkContractOnERC721Received(address,address,uint256,bytes) (
    ↪ contracts/txtest.sol#564-581) potentially used before declaration
    ↪ : reason.length == 0 (contracts/txtest.sol#573)
Variable 'ERC721A._checkContractOnERC721Received(address,address,uint256
    ↪ ,bytes).reason (contracts/txtest.sol#572)' in ERC721A.
    ↪ _checkContractOnERC721Received(address,address,uint256,bytes) (
    ↪ contracts/txtest.sol#564-581) potentially used before declaration
    ↪ : revert(uint256,uint256)(32 + reason,mload(uint256)(reason)) (
    ↪ contracts/txtest.sol#577)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #pre-declaration-usage-of-local-variables

Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/
    ↪ contracts/utils/Address.sol#201-221) uses assembly
 - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol
    ↪ #213-216)
ERC721A._checkContractOnERC721Received(address,address,uint256,bytes) (
    ↪ contracts/txtest.sol#564-581) uses assembly
 - INLINE ASM (contracts/txtest.sol#576-578)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #assembly-usage

Different versions of Solidity are used:
 - Version used: ['^0.8.0', '^0.8.1', '^0.8.4']
 - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol
    ↪ #4)
 - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/
    ↪ IERC721Receiver.sol#4)
```

```
 - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/
    ↪ IERC721Metadata.sol#4)
 - ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
 - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
 - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4)
 - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/
    ↪ ERC165.sol#4)
 - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/
    ↪ IERC165.sol#4)
 - ^0.8.4 (contracts/txtest.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #different-pragma-directives-are-used

ERC721A._burn(uint256) (contracts/txtest.sol#472-474) is never used and
   ↪ should be removed
ERC721A._burn(uint256,bool) (contracts/txtest.sol#486-539) is never used
   ↪  and should be removed
ERC721A._getAux(address) (contracts/txtest.sol#161-163) is never used
   ↪ and should be removed
ERC721A._mint(address,uint256,bytes,bool) (contracts/txtest.sol#365-407)
   ↪  is never used and should be removed
ERC721A._numberBurned(address) (contracts/txtest.sol#154-156) is never
   ↪ used and should be removed
ERC721A._numberMinted(address) (contracts/txtest.sol#147-149) is never
   ↪ used and should be removed
ERC721A._safeMint(address,uint256) (contracts/txtest.sol#333-335) is
   ↪ never used and should be removed
ERC721A._safeMint(address,uint256,bytes) (contracts/txtest.sol#347-353)
   ↪ is never used and should be removed
ERC721A._setAux(address,uint64) (contracts/txtest.sol#169-171) is never
   ↪ used and should be removed
ERC721A._totalMinted() (contracts/txtest.sol#118-124) is never used and
   ↪ should be removed
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/
    ↪ IERC721.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/
    ↪ IERC721Receiver.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/
    ↪ extensions/IERC721Metadata.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address
    ↪ .sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context
    ↪ .sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings
    ↪ .sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/
    ↪ introspection/ERC165.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/
    ↪ introspection/IERC165.sol#4) allows old versions
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (node_modules/
    ↪ @openzeppelin/contracts/utils/Address.sol#60-65):
 - (success) = recipient.call{value: amount}() (node_modules/
     ↪ @openzeppelin/contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,uint256,
    ↪ string) (node_modules/@openzeppelin/contracts/utils/Address.sol
    ↪ #128-139):
 - (success,returndata) = target.call{value: value}(data) (node_modules/
     ↪ @openzeppelin/contracts/utils/Address.sol#137)
Low level call in Address.functionStaticCall(address,bytes,string) (
    ↪ node_modules/@openzeppelin/contracts/utils/Address.sol#157-166):
```

```
 - (success,returndata) = target.staticcall(data) (node_modules/
    ↪ @openzeppelin/contracts/utils/Address.sol#164)
Low level call in Address.functionDelegateCall(address,bytes,string) (
   ↪ node_modules/@openzeppelin/contracts/utils/Address.sol#184-193):
 - (success,returndata) = target.delegatecall(data) (node_modules/
    ↪ @openzeppelin/contracts/utils/Address.sol#191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #low-level-calls

Parameter ERC721A.safeTransferFrom(address,address,uint256,bytes)._data
   ↪ (contracts/txtest.sol#313) is not in mixedCase
Variable ERC721A._currentIndex (contracts/txtest.sol#67) is not in
   ↪ mixedCase
Variable ERC721A._burnCounter (contracts/txtest.sol#70) is not in
   ↪ mixedCase
Variable ERC721A._ownerships (contracts/txtest.sol#80) is not in
   ↪ mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #conformance-to-solidity-naming-conventions

totalSupply() should be declared external:
 - ERC721A.totalSupply() (contracts/txtest.sol#107-113)
balanceOf(address) should be declared external:
 - ERC721A.balanceOf(address) (contracts/txtest.sol#139-142)
name() should be declared external:
 - ERC721A.name() (contracts/txtest.sol#214-216)
symbol() should be declared external:
 - ERC721A.symbol() (contracts/txtest.sol#221-223)
tokenURI(uint256) should be declared external:
 - ERC721A.tokenURI(uint256) (contracts/txtest.sol#228-233)
approve(address,uint256) should be declared external:
 - ERC721A.approve(address,uint256) (contracts/txtest.sol#247-256)
setApprovalForAll(address,bool) should be declared external:
```

```
 - ERC721A.setApprovalForAll(address,bool) (contracts/txtest.sol
    ↪ #270-275)
transferFrom(address,address,uint256) should be declared external:
 - ERC721A.transferFrom(address,address,uint256) (contracts/txtest.sol
    ↪ #287-293)
safeTransferFrom(address,address,uint256) should be declared external:
 - ERC721A.safeTransferFrom(address,address,uint256) (contracts/txtest.
    ↪ sol#298-304)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #public-function-that-could-be-declared-external
. analyzed (9 contracts with 78 detectors), 44 result(s) found
```

## Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

# 6 Conclusion

In this audit, we examined the design and implementation of Tribex contract and discovered several issues of varying severity. Tribex team acknowledged all the issues by classifying them a risk with low-probability of occurrence. Shellboxes' auditors advised Tribex Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.


For a Contract Audit, contact us at contact@shellboxes.com