TOURNIVERSE

Smart Contract Audit
Final Report

June 13, 2021

This 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.

Introduction

1. About Tourniverse

Tourniverse is a new cryptocurrency that focuses on the online gaming community. We want to create an online platform focused on providing a competitive, social, and safe atmosphere where gamers can compete against each other.

Playing ranked games is often considered as the most satisfying gamemode as you get rewarded for skill. However, playing competitively and making money is not for everyone as only a select few pro-gamers get the opportunity to compete for prize pools. Tourniverse's main goal is to make competing for prize pools more accessible for every gamer.

The development of the platform is divided into different chapters. As we are a community driven platform, we expect a certain amount of participation, activity and feedback to continue further development. This way, the team can optimize the platform to fit the desires and needs of our community!

Visit https://tourniverse.net/ to know more about.

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

No documentation was provided by the team for the purpose of the audit.

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: Tourniverse
- Languages: Solidity(Smart contract), Javascript(Unit Testing)
- Github commit hash/Smart Contract Address for audit:
    - Link: https://bscscan.com/token/0xF3dcCb92A98D0196a270FbA7a1D0125c89313e9b
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

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:

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	-	4(Acknowledged)	3(Acknowledged)

This 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.

Medium severity issues

1. **Loops are extremely costly**
   Line no -955, 1047
   Description:
   The Tourniverse contract has some for loops in the contract that include state variables like .length of a non-memory array, in the condition of the for loops.

   As a result, these state variables consume a lot more extra gas for every iteration of the for loop.
   The following function includes such loops at the above-mentioned lines:
   - includeInReward
   - _getCurrentSupply

```
1047        for (uint256 i = 0; i < _excluded.length; i++) {
1048            if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) re
1049            rSupply = rSupply.sub(_rOwned[_excluded[i]]);
1050            tSupply = tSupply.sub(_tOwned[_excluded[i]]);
1051        }
```

Recommendation:
It's quite effective to use a local variable instead of a state variable like .length in a loop. This will be a significant step in optimizing gas usage.

For instance,
local_variable = excluded.length;
 for (uint256 index = 0; index < local_variable; index++) {
    if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal, _tTotal);
            rSupply = rSupply.sub(_rOwned[_excluded[i]]);
            tSupply = tSupply.sub(_tOwned[_excluded[i]]);
}
}

Acknowledged(June 13th 2021): Tourniverse team has acknowledged the issue.

Note by the team:

includeInReward() will never be used again since the ownership of the contract is renounced. Optimizing this function is therefore regarded as non-valuable. The only time

this function was used was during the creation of the contract by the development team. Gas usage is therefore a non-issue.
While _getCurrentSupply() can be used within the contract, it will not matter if the contract uses .length within the for loop or before that much, calculation costs will not matter as much.

2. **Violation of Check_Effects_Interaction Pattern in the Withdraw function**
   Line no - 837-856
   Description:
   As per the current design of the Tourniverse contract, the constructor of the contract includes an external call. However, this external call is made before updating the imperative state variable of the contract.
   This approach violates the Check-Effects Interaction pattern.

```
844         IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);
845          // Create a uniswap pair for this new token
846         uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
847             .createPair(address(this), _uniswapV2Router.WETH());
848
849         // set the rest of the contract variables
850         uniswapV2Router = _uniswapV2Router;
851         //exclude owner and this contract from fee
852         _isExcludedFromFee[owner()] = true;
853         _isExcludedFromFee[address(this)] = true;
```

   Therefore, as per the Solidity Guidelines, any modification of the state variables in the base contract must be performed before executing the external call.

   The following function in the contract updates the state variables after making an external call at the lines mentioned below:

   - constructor

   Recommendation:
   Check Effects Interaction Pattern must be followed while implementing external calls in a function.

   Acknowledged(June 13th 2021): Tourniverse team has acknowledged the issue.

   Note by the team:
   This function uses a unicrypt pair (PCS LQ) which is confirmed and safe. Check effects are considered in all other functions that could potentially harm the contract. This specific part of the contract does not.

3. **Absence of Input Validations**
   Line no - 985, 989, 993
   Description:

The Tourniverse contract includes quite a few functions that update some crucial state variables of the contract.
However, no input validation is included in any of those functions. This might lead to an unwanted scenario where an invalid or wrong argument is passed to the function that could badly affect the expected behavior of the contract.

```
985     function setTaxFeePercent(uint256 taxFee) external onlyOwne
986         _taxFee = taxFee;
987     }
988
989     function setCharityFeePercent(uint256 charityFee) external
990         _charityFee = charityFee;
991     }
992
993     function setLiquidityFeePercent(uint256 liquidityFee) exter
994         _liquidityFee = liquidityFee;
995     }
```

Recommendation:
Arguments passed to such functions must be validated before being used to update the State variable.

Acknowledged(June 13th 2021): Tourniverse team has acknowledged the issue.

Note by the team:
Since the team will renounce ownership after having put the values ALREADY. These functions will never be used again in the contract. Therefore a validation of input was NOT implemented within the contract.

4. **No Events emitted after imperative State Variable modification**
   Line no -985, 989, 993

   Description:
   Functions that update an imperative arithmetic state variable contract should emit an event after the updation.
   The following functions modify some crucial arithmetic parameters like _taxFee, _charityFee etc in the Tourniverse contract but don't emit any event after that:

   - setTaxFeePercent()
   - setCharityFeePercent()
   - setLiquidityFeePercent()

Since there is no event emitted on updating these variables, it might be difficult to track it off chain.

Recommendation:
An event should be fired after changing crucial arithmetic state variables.

Acknowledged(June 13th 2021): Tourniverse team has acknowledged the issue.

Note by the team:
The team will renounce ownership. These functions can therefore NEVER be used, and are not relevant to the contract anymore.

Low severity issues

1. **Absence of Zero Address Validation**
   Line no-
   Description:
   The Tourniverse contract includes quite a few functions that updates some of the imperative addresses in the contract like uniswapV2Pair etc.
   However, during the automated testing of the contact it was found that no Zero Address Validation is implemented on the following functions while updating the address state variables of the contract:
   - setUniswapPair

   Recommendation:
   A require statement should be included in such functions to ensure no zero address is passed in the arguments.

   Acknowledged(June 13th 2021): Tourniverse team has acknowledged the issue.

   Note by the team:
   Since the ownership will be renounced NO (0) functions can be used where imperative addresses will be used. Therefore validations are NOT required and are disregarded within the contract for optimization purposes.

2. **External Visibility should be preferred**
   Description:
   Those functions that are never called throughout the contract should be marked as external visibility instead of public visibility.

This will effectively result in Gas Optimization as well.

Therefore, the following function must be marked as external within the contract:

- isExcludedFromReward()
- totalFees()
- deliver()
- reflectionFromToken()
- excludeFromReward()
- excludeFromFee()
- includeInFee()
- setSwapAndLiquifyEnabled()
- isExcludedFromFee()

Recommendation:
If the PUBLIC visibility of the above-mentioned functions is not intended, then the EXTERNAL Visibility keyword should be preferred.

Acknowledged(June 13th 2021): Tourniverse team has acknowledged the issue.

Note by the team:

Tourniverse is FULL transparent, therefore this was intentional.

3. **Contract includes Hardcoded Addresses**
   Line no - 451-457
   Description:
   Keeping in mind the immutable nature of smart contracts, it is not considered a better practise to hardcode any address in the contract before deployment.
   However, the contract does include some hardcoded addresses in the above-mentioned lines.

```
794
795     address private _charityWalletAddress = 0x1aeb122935fAEFC224f9AA70553601D537753695;
796
```

Recommendation:
By including hardcoded addresses in the contract, it would be an effective approach to initialize those addresses within the constructors at the time of deployment.

Acknowledged(June 13th 2021): Tourniverse team has acknowledged the issue.

Note by the team:

Tourniverse is FULL transparent, therefore this was intentional.

Recommendations

1. **Coding Style Issues in the Contract**
   Explanation:
   Code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

```
Parameter Tourniverse.setSwapAndLiquifyEnabled(bool)._enabled (contracts/TRNI.sol#1003) is not in mixedCase
Parameter Tourniverse.calculateTaxFee(uint256)._amount (contracts/TRNI.sol#1072) is not in mixedCase
Parameter Tourniverse.calculateCharityFee(uint256)._amount (contracts/TRNI.sol#1078) is not in mixedCase
Parameter Tourniverse.calculateLiquidityFee(uint256)._amount (contracts/TRNI.sol#1084) is not in mixedCase
Variable Tourniverse._taxFee (contracts/TRNI.sol#806) is not in mixedCase
Variable Tourniverse._charityFee (contracts/TRNI.sol#809) is not in mixedCase
Variable Tourniverse._liquidityFee (contracts/TRNI.sol#811) is not in mixedCase
Variable Tourniverse._maxTxAmount (contracts/TRNI.sol#820) is not in mixedCase
```

   During the automated testing, it was found that the Tourniverse contract had quite a few code style issues.

   Recommendation:
   Therefore, it is highly recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

2. **NatSpec Annotations must be included**
   Description:
   The smart contracts do not include the NatSpec annotations adequately.

   Recommendation:
   Cover by NatSpec all Contract methods.

3. **Commented codes must be wiped-out before deployment**
   Line no - 944
   Description:
   The Tourniverse contract includes quite a few commented codes in the excludeFromReward .
   This badly affects the readability of the code

```
ftrace | funcSig
943     function excludeFromReward(address account) public onlyOwner() {
944         // require(account != 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We can not exclude Uniswap router.');
945         require(!_isExcluded[account], "Account is already excluded");
```

   Recommendation:

This 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.

If these instances of code are not required in the current version of the contract, then the commented codes must be removed before deployment.

4. **Constant declaration should be preferred**
   Line no - 795,798,802,803,804
   Description:
   State variables that are not supposed to change throughout the contract should be declared as constant.

   Recommendation:
   The following state variables need to be declared as constant, unless the current contract design is intended.

   - _charityWalletAddress at Line 795
   - _decimals at Line 804
   - _name at Line 802
   - _symbol at Line 803
   - _tTotal at Line 798

5. **Too many Digits used**
   Line no - 798,820,821
   Description:
   The above-mentioned lines have a large number of digits that reduces the readability of the code.
   - _tTotal at Line 798
   - _maxTxAmount at Line 820
   - numTokensSellToAddToLiquidity at Line 821

   Recommendation:
   Ether Suffix could be used to symbolize the 10^18 zeros.

Automated Audit

```
deliver(uint256) should be declared external:
        - Tourniverse.deliver(uint256) (contracts/TRNI.sol#917-924)
reflectionFromToken(uint256,bool) should be declared external:
        - Tourniverse.reflectionFromToken(uint256,bool) (contracts/TRNI.sol#926-935)
excludeFromReward(address) should be declared external:
        - Tourniverse.excludeFromReward(address) (contracts/TRNI.sol#943-951)
excludeFromFee(address) should be declared external:
        - Tourniverse.excludeFromFee(address) (contracts/TRNI.sol#977-979)
includeInFee(address) should be declared external:
        - Tourniverse.includeInFee(address) (contracts/TRNI.sol#981-983)
setSwapAndLiquifyEnabled(bool) should be declared external:
        - Tourniverse.setSwapAndLiquifyEnabled(bool) (contracts/TRNI.sol#1003-1006)
isExcludedFromFee(address) should be declared external:
```

```
Tourniverse.allowance(address,address).owner (contracts/TRNI.sol#884) shadows:
        - Ownable.owner() (contracts/TRNI.sol#545-547) (function)
Tourniverse._approve(address,address,uint256).owner (contracts/TRNI.sol#1113) shadows:
        - Ownable.owner() (contracts/TRNI.sol#545-547) (function)
```

```
Tourniverse._rTotal (contracts/TRNI.sol#799) is set pre-construction with a non-constant function or state variable:
        - (MAX - (MAX % _tTotal))
Tourniverse._previousTaxFee (contracts/TRNI.sol#807) is set pre-construction with a non-constant function or state variable:
        - _taxFee
Tourniverse._previousCharityFee (contracts/TRNI.sol#810) is set pre-construction with a non-constant function or state variable:
        - _charityFee
Tourniverse._previousLiquidityFee (contracts/TRNI.sol#812) is set pre-construction with a non-constant function or state variable:
        - _liquidityFee
```

```
Tourniverse._charityWalletAddress (contracts/TRNI.sol#795) should be constant
Tourniverse._decimals (contracts/TRNI.sol#804) should be constant
Tourniverse._name (contracts/TRNI.sol#802) should be constant
Tourniverse._symbol (contracts/TRNI.sol#803) should be constant
Tourniverse._tTotal (contracts/TRNI.sol#798) should be constant
```

Concluding Remarks

While conducting the audits of the Tourniverse smart contract, it was observed that the contracts contain Medium, and Low severity issues, along with several areas of recommendations.

Our auditors suggest that High, Medium, Low severity issues should be resolved by the developers. Resolving the areas of recommendations are up to the team's discretion. 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 Tourniverse 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 Pvt Ltd.

This 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.