Ethernity

Satoshi Staking

Smart Contract Audit
Final Report

May 10, 2022

Introduction

1. About Ethernity

Ethernity is a Decentralized Application (DAPP) Platform that allows artists to create and auction artwork inspired and backed by celebrities for charity.
The concept behind Ethernity is mutually beneficial for all actors involved:

1. **Public Figure:** by making it easier to create, store, back, and sell the artworks.
2. **Charity:** by getting 100% of the first sale proceeds (minus exchange fees). And the auction format maximizes the artwork value (increasing the charity's benefits) without the need of a promoter, leveraging the emotions that a bidding war involves.
3. **Collector:** by providing them with an easy, democratized platform to bid on these pieces of authentic digital art where they can thereafter take bids and auction their acquired artwork.

   With ERN tokens collectors can acquire Ethernity's exclusive authenticated NFTs as a payment method and also yield farming rewards. Part of the sales proceeds goes to charity.

Visit https://ethernity.io/ to know more about it.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provides 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, and dydx.

The team has been able to secure 105+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-ups 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 Ethernity team has provided the following doc for the purpose of audit:

1. https://github.com/ethernitychain/satoshi-staking/tree/dev#readme
2. https://ethernity.cloud/whitepaper/ETHERNITY_whitepaper.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: Ethernity
- Contracts Name: SatoshiStaking.sol
- Languages: Solidity(Smart contract), Typescript (Unit Testing)
- Github commits for initial audit: 39b9e2ce3435e61d970ca9aea0ab3de3737b9d45
- Github commits for final audit: 4789eb66aa4a3585fab12a59d00ad9eea01d95b5
- 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 Reference

Every issue in this report were 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	1	3	3

High Severity Issues

1. **Secret information should not be placed on-chain**
   Line no: - 57

   Description:
   The current contract includes a state variable, i.e., **magicData**.

   As per the inline documentation found in the contract, this state variable is supposed to be secret data that shall be used for preventing direct NFT Transfers.

```
154
155     /*
156      *With this function, users will be able to stake both ERC721 and 1155 types .
157      *magicData: our secret variable, we'll use it for preventing direct nft transfers.
158      */
```

   However, the state variable is marked as PUBLIC in the current contract design.

```
70      // address of the admin
71      address admin;
72      // a data for controlling direct transfer
73      bytes private magicData;
```

   Most importantly, it must be kept in mind that every state of a contract is basically publicly visible, even if a state variable is marked as **private**. Therefore, it's not considered a secure practice to store any sensitive data on-chain.

   Recommendation:
   Sensitive or secret information must not be put on-chain since it's always publicly visible.

   Note:
   Although the magicData bytes state variable is now made private, it doesn't ensure that data is not visible to users of the protocol. If this is supposed to be very sensitive data that perform imperative functions, it should be kept off-chain as on-chain state variables are accessible to users even if they are marked as private. Read more here

   Acknowledged (May 10th, 2022).

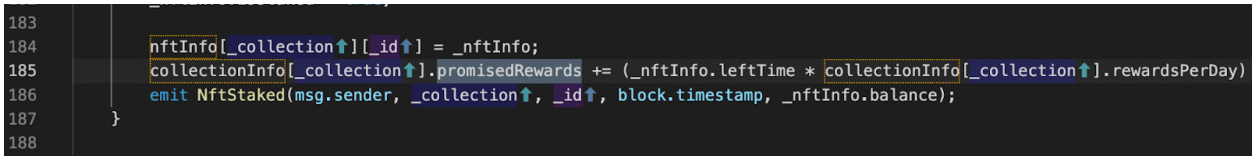
   Note by team:

   The critical vulnerability is not a vulnerability at all, works just as intended. It is just a 2FA, preventing users to do tx by mistake, without 2FA code.

Medium Severity Issues

1. **Promised rewards can be manipulated within the lifetime of an NFT, thus leading to no claimable rewards at all**
   Line no: 293-304

   Description:
   As per the current architecture of the contract, the **promisedRewards** of a specific collection address is updated right at the time of the staking in the stakeSingleNFT() function. Additionally, this promised rewards amount is derived from the amount of time left as well as the amount of the per-day reward for that collection.

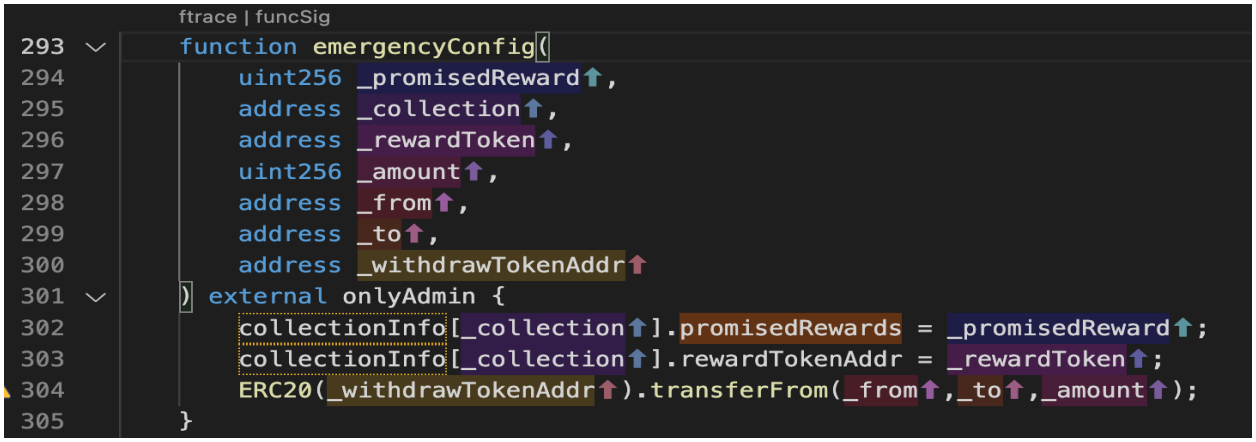


```
183
184         nftInfo[_collection⬆][_id⬆] = _nftInfo;
185         collectionInfo[_collection⬆].promisedRewards += (_nftInfo.leftTime * collectionInfo[_collection⬆].rewardsPerDay)
186         emit NftStaked(msg.sender, _collection⬆, _id⬆, block.timestamp, _nftInfo.balance);
187     }
188
```

   The **promisedRewards** of a collection is an imperative state since it plays a major role whenever a user tries to claim their rewards through the **claimRewards**() function. (Line 215 at claimRewards function)

```
211         address tokenAdd = collectionInfo[_collection⬆].rewardTokenAddr; //*********** */
212         uint256 rewardTokenBalance = ERC20(tokenAdd).balanceOf(address(this));
213         require(rewardTokenBalance >= reward, "There is no enough reward token to give you!
214
215         collectionInfo[_collection⬆].promisedRewards -= reward;
216
```

   However, the contract also includes an **emergencyConfig**() function that places enormous power in the hands of the admin as it allows the admin to increase or decrease the promised rewards value for any collection at any given time.



```
ftrace | funcSig
293     function emergencyConfig(
294         uint256 _promisedReward⬆,
295         address _collection⬆,
296         address _rewardToken⬆,
297         uint256 _amount⬆,
298         address _from⬆,
299         address _to⬆,
300         address _withdrawTokenAddr⬆
301     ) external onlyAdmin {
302         collectionInfo[_collection⬆].promisedRewards = _promisedReward⬆;
303         collectionInfo[_collection⬆].rewardTokenAddr = _rewardToken⬆;
304         ERC20(_withdrawTokenAddr⬆).transferFrom(_from⬆,_to⬆,_amount⬆);
305     }
```

Moreover, during the review, it was found that the function doesn't include any input validations which technically allows the admin to set the promised rewards for a collection, directly to zero. This shall lead to a very unwanted scenario where the users will be unable to claim any rewards at all.

Recommendation:
The contract design for the above-mentioned functions, specifically the **emergencyConfig**() function, must be modified adequately.
If the above-mentioned scenario is not intentional, then the **emergencyConfig** function could be modified to execute in a way that reduces the chances of any inadequate manipulation of the promised rewards for any given collection.

For instance:

a. The admin could only be able to update promised rewards after or before a specific point during the lifetime of an NFT.
b. The function could be designed to protect the promised rewards to be completely zero and avoid the scenario of zero claimable rewards for users.

Amended (May 10th, 2022): The issue was fixed by the **Ethernity** team and is no longer present in commit 4789eb66aa4a3585fab12a59d00ad9eea01d95b5.

2. **Equality of array lengths is not validated before function execution**
Line no: 189-194

Description:
The **stakeBatchNFT()** function allows users to stake a batch of nft by passing an array of the collection addresses and ids.
However, during the review, it was found that the length of arrays being passed as arguments isn't validated to be equal before the actual function execution.

For functions that include more than one array argument must ensure that arrays with similar lengths have been passed as it might lead to an error while iterating over these arrays in the for loop of the function.

Recommendation:
A **require** statement must be included to ensure that arrays with valid lengths are passed to the function.

Amended (May 10th, 2022): The issue was fixed by the **Ethernity** team and is no longer present in commit 4789eb66aa4a3585fab12a59d00ad9eea01d95b5.

3. **Multiplication is being performed on the result of Division**
   Line no - 113 to 117

   Description:
   During the code review of the **SatoshiContract** contract, it was found that the **getNFTInformation()** function in the contract is performing multiplication on the result of a Division, within its body.

```
112
113     _leftDays = leftTimeInSeconds / 1 days;
114     uint256 leftHoursInSeconds = leftTimeInSeconds - (_leftDays * 1 days);
115     _leftHours = leftHoursInSeconds / 3600;
116     uint256 leftMinutesInSeconds = leftHoursInSeconds - (_leftHours * 3600);
117     _leftMinutes = leftMinutesInSeconds / 60;
118
```

   Integer Divisions in Solidity might truncate. Moreover, performing division before multiplication might lead to a loss of precision.

   Recommendation:
   Solidity doesn't encourage arithmetic operations that involve division before multiplication. Therefore the above-mentioned function should be tested once again for all the corner cases and redesigned if they do not lead to the expected results.

   Amended (May 10th, 2022): The issue was fixed by the **Ethernity** team and is no longer present in commit 4789eb66aa4a3585fab12a59d00ad9eea01d95b5.

Low Severity Issues

1. **Violation of Check_Effects_Interaction Pattern in the Withdraw function**

   Description:
   The **SatoshiStaking** contract includes a few functions that update some of the very imperative state variables of the contract after the external calls are being made.

   An external call within a function technically shifts the control flow of the contract to another contract for a particular period of time. 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 functions in the contract update the state variables after making an external call at the lines mentioned below:

   - **fundCollection() at Line 271**
   - **unstake() at Line 243-245**

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

Amended (May 10th, 2022): The issue was fixed by the **Ethernity** team and is no longer present in commit 4789eb66aa4a3585fab12a59d00ad9eea01d95b5.

2. **Absence of Zero Address Validation**

Description:
The constructor of the contract updates the imperative admin address of the contract.
However, during the automated testing of the contract, it was found that no Zero Address Validation is implemented on the function to ensure that only valid addresses are passed to it.

Recommendation:
Zero Address validations should be implemented.

Amended (May 10th, 2022): The issue was fixed by the **Ethernity** team and is no longer present in commit 4789eb66aa4a3585fab12a59d00ad9eea01d95b5.

3. **Unwanted state variables could be removed**
Line no: 59, 60

Description:
The contract currently includes a few state variables that indicate no significant use throughout the contract execution.

```
58
59      uint256 public TESTRESULT;
60      uint256 public TEST2RESULT;
61
```

Moreover, it appears that these variables might have been implemented for testing purposes but were never removed. This increases unnecessary gas usage in functions as well.

Recommendation:
Unwanted state variables or functions should be removed from the contract to optimize gas and enhance performance.

Amended (May 10th, 2022): The issue was fixed by the **Ethernity** team and is no longer present in commit 4789eb66aa4a3585fab12a59d00ad9eea01d95b5.

Recommendation / Informational

1. **Exactly similar error messages were found**
   Line no: 135, 136

   Description:
   The **addCollection**() function in the contract, includes 2 crucial require statements at the above-mentioned lines that check whether or not the given address of the **_collection** is a contract and to ensure that the reward token address is a contract address.

```
134
135     require(Controller.isContract(_collection), "Given address does not belong to any contract!");
136     require(Controller.isContract(_collecInfo.rewardTokenAddr), "Given address does not belong to any contract!");
137
```

   However, for both of these require statements, the exact same error messages have been used which might lead to an ambiguous situation where the specific source of error will be harder to find.

   Moreover, having similar error messages for two different require statements also affects the code readability.

   Recommendation:
   Error messages should be unique to each **require** statement and should be stated clearly.

   Amended (May 10th, 2022): The issue was fixed by the **Ethernity** team and is no longer present in commit 4789eb66aa4a3585fab12a59d00ad9eea01d95b5.

2. **Inadequate Test cases found**

   Description:
   The test cases attached with the contract don't seem to be adequate enough because
   a. The current set of test cases doesn't cover the entire contract and every function
   b. The test cases provided currently fail in particular instances specifically during the calculation of claimable rewards.

   Recommendation:
   Test cases must be improved and modified to cover all function and imperative corner cases.

   Amended (May 10th, 2022): The issue was fixed by the **Ethernity** team and is no longer present in commit 4789eb66aa4a3585fab12a59d00ad9eea01d95b5.

3. **NatSpec Annotations must be included**

Description:
The smart contracts do not include the NatSpec annotations adequately.

Recommendation:
Cover by NatSpec all Contract methods.

Amended (May 10th, 2022): The issue was fixed by the **Ethernity** team and is no longer present in commit 4789eb66aa4a3585fab12a59d00ad9eea01d95b5.

4. **Coding Style Issues in the Contract**

Description:
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 SatoshiStaking.computeReward(address,uint256)._collection (contracts/SatoshiStaking.sol#77) is not in mixedCase
Parameter SatoshiStaking.computeReward(address,uint256)._id (contracts/SatoshiStaking.sol#77) is not in mixedCase
Parameter SatoshiStaking.getNFTInformation(address,uint256)._collection (contracts/SatoshiStaking.sol#91) is not in mixedCase
Parameter SatoshiStaking.getNFTInformation(address,uint256)._id (contracts/SatoshiStaking.sol#91) is not in mixedCase
Parameter SatoshiStaking.addCollection(address,SatoshiStaking.Collection)._collection (contracts/SatoshiStaking.sol#130) is not in
mixedCase
Parameter SatoshiStaking.addCollection(address,SatoshiStaking.Collection)._collecInfo (contracts/SatoshiStaking.sol#130) is not in
mixedCase
Parameter SatoshiStaking.removeCollection(address,bool)._collection (contracts/SatoshiStaking.sol#150) is not in mixedCase
Parameter SatoshiStaking.removeCollection(address,bool)._is721 (contracts/SatoshiStaking.sol#150) is not in mixedCase
Parameter SatoshiStaking.stakeSingleNFT(address,uint256)._collection (contracts/SatoshiStaking.sol#159) is not in mixedCase
Parameter SatoshiStaking.stakeSingleNFT(address,uint256)._id (contracts/SatoshiStaking.sol#159) is not in mixedCase
Parameter SatoshiStaking.stakeBatchNFT(address[],uint256[])._collections (contracts/SatoshiStaking.sol#189) is not in mixedCase
Parameter SatoshiStaking.stakeBatchNFT(address[],uint256[])._id (contracts/SatoshiStaking.sol#189) is not in mixedCase
Parameter SatoshiStaking.claimReward(address,uint256)._collection (contracts/SatoshiStaking.sol#196) is not in mixedCase
Parameter SatoshiStaking.claimReward(address,uint256)._id (contracts/SatoshiStaking.sol#196) is not in mixedCase
```

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

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

Amended (May 10th, 2022): The issue was fixed by the **Ethernity** team and is no longer present in commit 4789eb66aa4a3585fab12a59d00ad9eea01d95b5.

5. **Commented codes must be wiped-out before deployment**

Description:
The contract includes quite a few commented codes within the contract body.

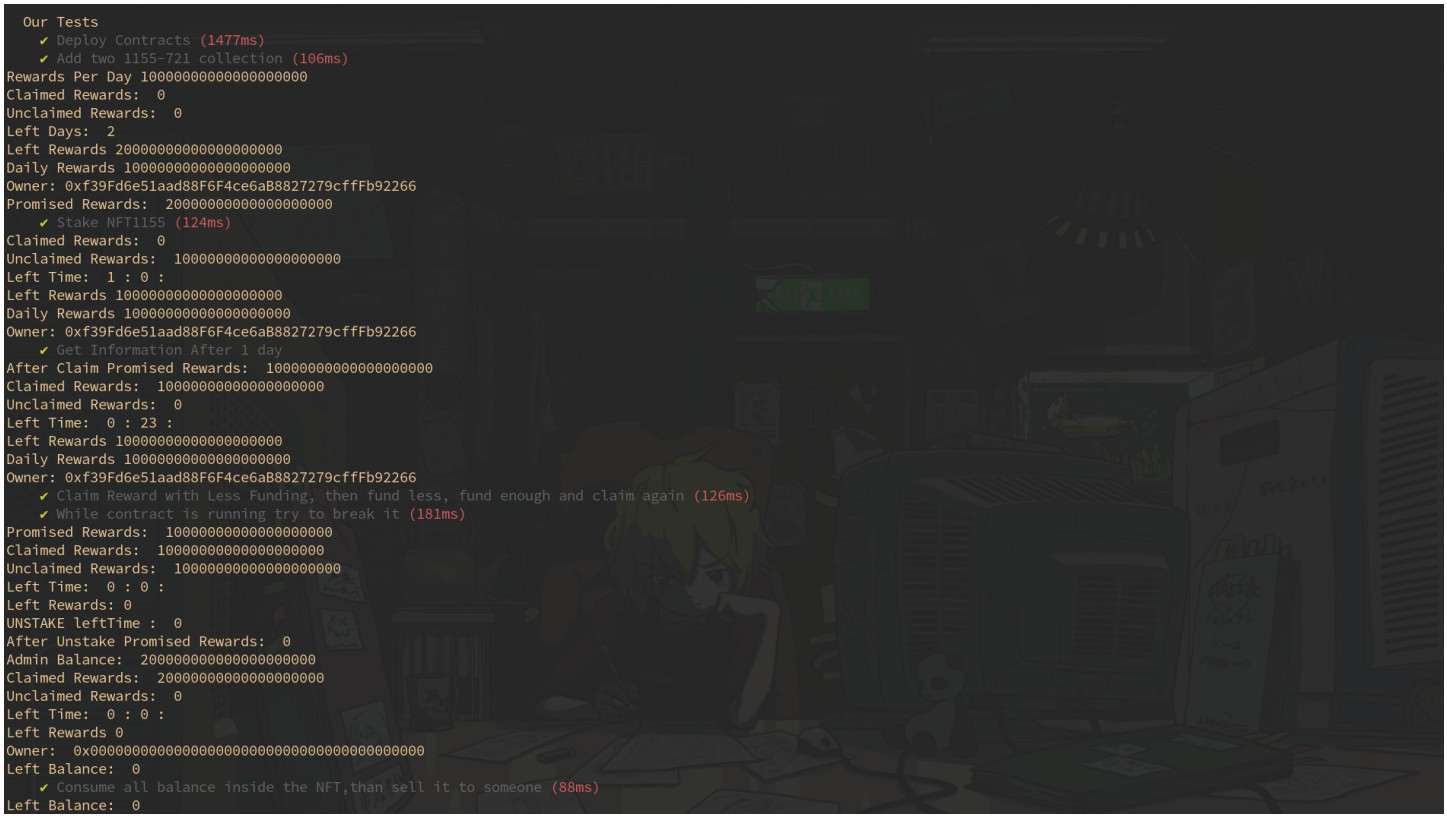
```
227
228         // require(
229         //     ERC20(nftInfo[_collection][_id].collec.rewardTokenAddr).transfer(msg.sender, reward),
230         //     "Couldn't transfer the amount!"
231         // );
232
```

This badly affects the readability of the code.

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

Amended (May 10th, 2022): The issue was fixed by the **Ethernity** team and is no longer present in commit 4789eb66aa4a3585fab12a59d00ad9eea01d95b5.

Unit Tests



```
  Our Tests
    ✔ Deploy Contracts (1477ms)
    ✔ Add two 1155-721 collection (106ms)
Rewards Per Day 1000000000000000000
Claimed Rewards:  0
Unclaimed Rewards:  0
Left Days:  2
Left Rewards 2000000000000000000
Daily Rewards 1000000000000000000
Owner: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
Promised Rewards:  2000000000000000000
    ✔ Stake NFT1155 (124ms)
Claimed Rewards:  0
Unclaimed Rewards:  1000000000000000000
Left Time:  1 : 0 :
Left Rewards 1000000000000000000
Daily Rewards 1000000000000000000
Owner: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
    ✔ Get Information After 1 day
After Claim Promised Rewards:  1000000000000000000
Claimed Rewards:  1000000000000000000
Unclaimed Rewards:  0
Left Time:  0 : 23 :
Left Rewards 1000000000000000000
Daily Rewards 1000000000000000000
Owner: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
    ✔ Claim Reward with Less Funding, then fund less, fund enough and claim again (126ms)
    ✔ While contract is running try to break it (181ms)
Promised Rewards:  1000000000000000000
Claimed Rewards:  1000000000000000000
Unclaimed Rewards:  1000000000000000000
Left Time:  0 : 0 :
Left Rewards: 0
UNSTAKE leftTime :  0
After Unstake Promised Rewards:  0
Admin Balance:  2000000000000000000
Claimed Rewards:  2000000000000000000
Unclaimed Rewards:  0
Left Time:  0 : 0 :
Left Rewards 0
Owner:  0x0000000000000000000000000000000000000000
Left Balance:  0
    ✔ Consume all balance inside the NFT,than sell it to someone (88ms)
Left Balance:  0
```

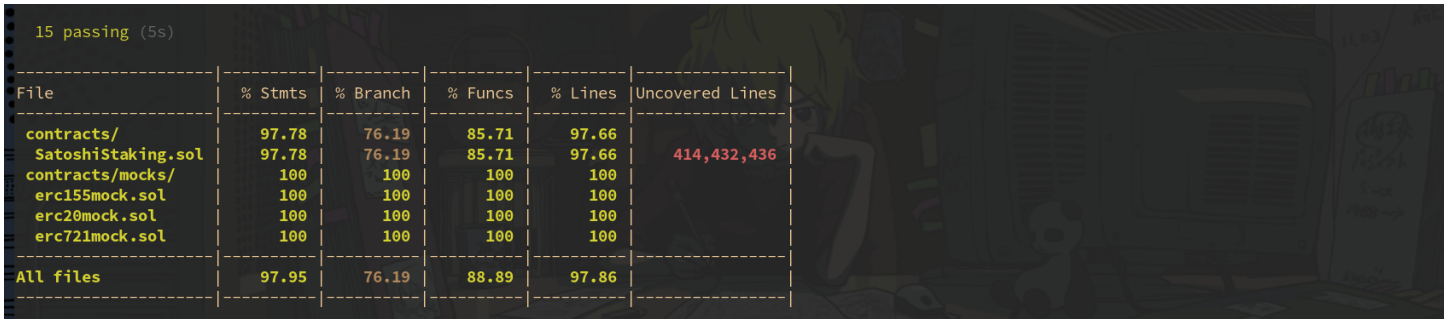
```
Left Rewards 0
Owner:  0x0000000000000000000000000000000000000000
Left Balance:  0
    ✔ Consume all balance inside the NFT,than sell it to someone (88ms)
Left Balance:  0
leftTime :  0
Bob Balance:  0
Claimed Rewards:  20000000000000000000
Unclaimed Rewards:  0
Left Time:  0 : 0 :
Left Rewards 0
Owner:  0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC
Bob Balance:  0
    ✔ Consume all balance inside the NFT,than sell it to someone (107ms)
Satoshi Reward Token Balance:  20000000000000000000
Admin before withdraw: 0
Promised Rewards:  0
Admin after withdraw: 20000000000000000000
    ✔ Try to withdraw funds (71ms)
    ✔ Mint NFT to people and test with them (243ms)
    ✔ Let them stake their NFTs (312ms)
Alice Balance:  12000000000000000000
Alice Balance:  12000000000000000000
Bob Balance:  0
Bob Balance:  0
Bob Balance:  0
Bob Balance:  0
    ✔ Mint NFT to people and test with them (756ms)
1-721 Left Balance:  0 Left Time:  0
2-721 Left Balance:  0 Left Time:  0
3-721 Left Balance:  20000000000000000000 Left Time:  172800
4-721 Left Balance:  20000000000000000000 Left Time:  172800
1-721-2 Left Balance:  0 Left Time:  0
2-721-2 Left Balance:  0 Left Time:  0
3-721-2 Left Balance:  20000000000000000000 Left Time:  172800
4-721-2 Left Balance:  0 Left Time:  0
1-1155 Left Balance:  0 Left Time:  0
2-1155 Left Balance:  0 Left Time:  0
3-1155 Left Balance:  20000000000000000000 Left Time:  172800
4-1155 Left Balance:  0 Left Time:  0
1-1155-2 Left Balance:  20000000000000000000 Left Time:  172800
2-1155-2 Left Balance:  20000000000000000000 Left Time:  172800
3-1155-2 Left Balance:  0 Left Time:  0
4-1155-2 Left Balance:  0 Left Time:  0
    ✔ Observe NFT balance and time-Some should be zero, some should have balance (199ms)
```

```
4-1155-2 Left Balance:  0 Left Time:  0
    ✔ Observe NFT balance and time-Some should be zero, some should have balance (199ms)
Left Reward:  20000000000000000000
After two days Unclaimed Reward:  20000000000000000000
Admin balance before withdrawing funds:  20000000000000000000
Satoshi balance before withdrawing funds:  99998600000000000000000000
Admin balance after withdrawing funds:  10000060000000000000000000
Satoshi balance after withdrawing funds:  0
    ✔ Unstake and withdraw funds (121ms)
Before r- All supported 1155 Collections:  [
  '0xDc64a140Aa3E981100a9becA4E685f962f0cF6C9',
  '0x5FC8d32690cc91D4c39d9d3abcBD16989F875707'
]
Before r- All supported 721 Collections:  [
  '0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512',
  '0x9fE46736679d2D9a65F0992F2272dE9f3c7fa6e0'
]
After r-All supported 1155 Collections:  []
After r- All supported 721 Collections:  []
    ✔ Unstake and withdraw funds (55ms)


  15 passing (4s)
```

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.

Test Coverage



```
  15 passing (5s)

----------------------|----------|----------|----------|----------|----------------|
File                  |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
----------------------|----------|----------|----------|----------|----------------|
 contracts/           |    97.78 |    76.19 |    85.71 |    97.66 |                |
  SatoshiStaking.sol  |    97.78 |    76.19 |    85.71 |    97.66 |    414,432,436 |
 contracts/mocks/     |      100 |      100 |      100 |      100 |                |
  erc155mock.sol      |      100 |      100 |      100 |      100 |                |
  erc20mock.sol       |      100 |      100 |      100 |      100 |                |
  erc721mock.sol      |      100 |      100 |      100 |      100 |                |
----------------------|----------|----------|----------|----------|----------------|
All files             |    97.95 |    76.19 |    88.89 |    97.86 |                |
----------------------|----------|----------|----------|----------|----------------|
```

Automated Audit Result

Maian

```
root@719b9408de70:/MAIAN/tool# python3 maian.py -b /share/flat.bytecode -c 0

======================================================================================================
[ ] Check if contract is SUICIDAL

[ ] Contract address   : 0xaFFECAFEAFfECaFEaFFecAfEAFfecAfEAffEcaFE
[ ] Contract bytecode  : 610497610053600b82828239805160001a607314610046577f...
[ ] Bytecode length    : 2516
[ ] Blockchain contract: False
[ ] Debug              : False

[-] The code does not contain SUICIDE instructions, hence it is not vulnerable
root@719b9408de70:/MAIAN/tool# python3 maian.py -b /share/flat.bytecode -c 1

======================================================================================================
[ ] Check if contract is PRODIGAL

[ ] Contract address   : 0xaFFECAFEAFfECaFEaFFecAfEAFfecAfEAffEcaFE
[ ] Contract bytecode  : 610497610053600b82828239805160001a607314610046577f...
[ ] Bytecode length    : 2516
[ ] Blockchain contract: False
[ ] Debug              : False
[+] The code does not have CALL/SUICIDE, hence it is not prodigal
root@719b9408de70:/MAIAN/tool# python3 maian.py -b /share/flat.bytecode -c 2

======================================================================================================
[ ] Check if contract is GREEDY

[ ] Contract address   : 0xaFFECAFEAFfECaFEaFFecAfEAFfecAfEAffEcaFE
[ ] Contract bytecode  : 610497610053600b82828239805160001a607314610046577f...
[ ] Bytecode length    : 2516
[ ] Debug              : False
[-] Contract can receive Ether

[-] No lock vulnerability found because the contract cannot receive Ether
```

Mythril

```
The analysis was completed successfully. No issues were detected.
```

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.

Slither

```
Compiled with solc
Number of lines: 1332 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 9 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 62
Number of low issues: 14
Number of medium issues: 7
Number of high issues: 2
ERCs: ERC721, ERC20, ERC165

+----------------+-------------+---------------+--------------------+--------------+--------------------+
|      Name      | # functions |      ERCS     |     ERC20 info     | Complex code |      Features      |
+----------------+-------------+---------------+--------------------+--------------+--------------------+
|     IERC20     |      6      |     ERC20     |     No Minting     |      No      |                    |
|                |             |               | Approve Race Cond. |              |                    |
|                |             |               |                    |              |                    |
|    IERC1155    |      7      |     ERC165    |                    |      No      |                    |
|    IERC721     |      10     | ERC165,ERC721 |                    |      No      |                    |
| EnumerableSet  |      24     |               |                    |      No      |      Assembly      |
|   Controller   |      3      |               |                    |      No      |      Assembly      |
| SatoshiStaking |      25     |     ERC165    |                    |      No      | Tokens interaction |
+----------------+-------------+---------------+--------------------+--------------+--------------------+
```

Concluding Remarks

While conducting the audits of the Ethernity smart contracts, it was observed that the contracts contain High, Medium, and Low severity issues.

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

Note: ***Ethernity team has Acknowledged/fixed the issues based on the auditor's recommendation. The Ethernity does not have any issues present in the 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 Ethernity 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