IMMUNEBYTES
Audits

WAM

SMART CONTRACT AUDIT FINAL REPORT

February 23, 2022

TOC

TABLE OF CONTENTS

Introduction

1. About WAM

WAM is a platform where people compete in tournaments to win crypto rewards in SWAM and NFTs. You can play using your mobile phone from Chrome, or Safari or you can download WAM.app from Google Play and Apple AppStore. Users on WAM can own games and tournaments to generate recurring revenue, developers can publish games and sell NFTs and marketers can promote tournaments to earn SWAM from them.

The concept of making money by playing hyper casual games based on skill is new in the gaming industry and WAM is the first platform in the world that let you do this. All you need in order to play on the WAM platform is a few minutes of your time and the desire to be the best. You can participate in tournaments all around the world, wherever you have internet access. Players don't have to be connected at the same time, and that is very good for players because each can compete whenever they find the time to do so.

Visit https://wam.app/ to know more about it.

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 125+ 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 WAM team has provided the following doc for the purpose of audit:

1. https://whitepaper.wam.app/

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: WAM
- Contracts Name: Staking.sol, Wam.sol
- Languages: Solidity(Smart contract), Typescript (Unit Testing)
- Audit Scope: https://github.com/Digitapeu/wam-staking-contract/
- Github commits for the initial audit: bbc89ab521b789f27dcd2dbbaa21fac58730ee9a
- Github commits for the final audit: 036250f542abfc59fd6f92483380749272d71f87
- 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:

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	5

Contract Name: Staking.sol & Wam.sol

High Severity Issues

No issues were found.

Medium severity issues

1. **OPERATOR_ROLE is initialized but never assigned to any specific address**
   Line no - 7

   Explanation:
   The **AccessLevel** contract uses **AccessControl** to assign specific roles in the contract to particular addresses. It initializes the **Operator Role** in the contract, at Line 7 to specify a new type of access level, particularly for the operators of the contract.

```
6  abstract contract AccessLevel is AccessControlUpgradeable {
7      bytes32 public constant OPERATOR_ROLE = keccak256("OPERATOR_ROLE");
8
9      function __AccessLevel_init(address owner) public initializer  {
10         __AccessControl_init();
11         _setupRole(DEFAULT_ADMIN_ROLE, owner);
12     }
13 }
14
```

   However, during the manual review, it was found that this role has not to be set up or assigned to any specific address and the Operator Role is never used throughout the contract.

   This will lead to a severe issue if any particular function, in the further upgrades, is only supposed to be accessed by operator roles while the operator role is never really assigned to any address.

   Recommendation:
   If the current contract design doesn't involve any significant use for the operator role, it can be removed from the contract. Otherwise, a specific address can be assigned an operator role in order to have the intended behavior of different access control levels in the contract.

   Amended (February 23rd, 2022): The issue was fixed by the **WAM** team and is no longer present in commit 036250f542abfc59fd6f92483380749272d71f87

2. **Multiplication is being performed on the result of Division**
Line no - 183-184, 187-188, 212-213, 246-247

Explanation:
During the automated testing of the **Staking** contract, it was found that some of the functions in the contract are performing multiplication on the result of a Division.
Integer Divisions in Solidity might truncate. Moreover, this performing division before multiplication might lead to a loss of precision.

While this might not lead to any severe issue, it is recommended to ensure adequate test cases have been included for this specific section to ensure it doesn't affect the intended behavior of the contracts.

The following functions involve division before multiplication in the mentioned lines:

- **getPossibleRewardsForUserStake()**
- **unstake(uint256)**
- **unstake(uint256,uint256)**

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

Amended (February 23rd, 2022): The issue was fixed by the **WAM** team and is no longer present in commit 036250f542abfc59fd6f92483380749272d71f87

Low severity issues

1. **External Visibility should be preferred**

   Explanation:
   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:

   - **getAllAddressStakes()**
   - **getLostPercentageNowForUserStake()**
   - **getLostAmountNowForUserStake()**
   - **getPossibleRewardsForUserStake()**

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

   Amended (February 23rd, 2022): The issue was fixed by the **WAM** team and is no longer present in commit 036250f542abfc59fd6f92483380749272d71f87

2. **No Events emitted after imperative State Variable modification**
   Line no -103-106

   Explanation:
   Functions that update an imperative arithmetic state variable contract should emit an event after the state modification.
   The **setMaxLoss()** function modifies a crucial arithmetic parameter, i.e., **maxLoss** in the Staking contract but doesn't emit any event:

   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.

   Amended (February 23rd, 2022): The issue was fixed by the **WAM** team and is no longer present in commit 036250f542abfc59fd6f92483380749272d71f87

3. **Unused State variable found in the contract**
Line no: 26

Explanation:
During the manual code review of the contract, it was found that the **stakingId** state variable is never used throughout the contract.

As per the current design of the contract, the **StakingInfo** struct already has a specific member named **id**, that can be used to track a particular staking id for a given address. Therefore, the **stakingId** doesn't seem to have adequate significance.

Recommendation:
It is recommended to remove any unused state variable in the contract.

Amended (February 23rd, 2022): The issue was fixed by the **WAM** team and is no longer present in commit 036250f542abfc59fd6f92483380749272d71f87

4. **Absence of Zero Address Validation**
Line no- 44-50, 112-114

Explanation:
The Staking contract includes a few functions that update some of the significant addresses in the contract like **stakingTokensAddress, communityAddress** 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:

- initialize()
- setCommunityAddress()

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

Amended (February 23rd, 2022): The issue was fixed by the **WAM** team and is no longer present in commit 036250f542abfc59fd6f92483380749272d71f87

5. **Event emissions can be improved in the contract**

Explanation:
As per the current design, the events **Stake & Unstake** in the contract do not include the amount parameter. This leads to a scenario where the amount of tokens being staked or unstaked is never emitted out, despite the fact that this could be imperative arithmetic data.

Moreover, considering the unstaking mechanism of the contract, it's possible for a user to unstake only a portion of his staked token. In such a scenario it's important to emit out the specific amount of tokens being unstaked.

Recommendation:
The events being emitted in the contract can be improved.

Amended (February 23rd, 2022): The issue was fixed by the **WAM** team and is no longer present in commit 036250f542abfc59fd6f92483380749272d71f87

Recommendations/Informational

1. **Code duplication found during review**

   Explanation:
   During the code review of the Staking contract, it was found that at a few instances, similar code logic was being repeated on multiple function calls which could have been abstracted out as a separate function and be reused.

   For instance, the calculation of **rewardPercentage, rewardForStaking, lossPercentage,** and **tokenLost** code duplication was found in the following function:

   - **getLostPercentageNowForUserStake()**
   - **getLostAmountNowForUserStake()**
   - **getPossibleRewardsForUserStake()**
   - **unstake()**

   Recommendation:
   Duplication of code should be avoided as it not just reduces code readability but also badly affects the gas optimization part aspect of the contract.

   Amended (February 23rd 2022): The issue was fixed by the **WAM** team and is no longer present in commit 036250f542abfc59fd6f92483380749272d71f87

2. **Absence of Pausable functionalities in the contract.**

   Explanation:
   During the code review it was found that the contract doesn't include any pausable feature that allows the owner to pause the contract in extreme conditions.

   Considering the fact that the contract has some publically accessible functions, including pausable features would ensure an additional layer of security for the contract.

   Recommendation:
   Unless the current design is intended, Pausable functionalities can be included in the contract as well.

Automated Audit Result

1. Staking:

```
Compiled with solc
Number of lines: 1223 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 12 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 8
Number of informational issues: 46
Number of low issues: 10
Number of medium issues: 6
Number of high issues: 1
ERCs: ERC165, ERC20

+----------------------+-------------+--------+-----------------------+--------------+---------------------+
|         Name         | # functions | ERCS   |       ERC20 info      | Complex code |       Features      |
+----------------------+-------------+--------+-----------------------+--------------+---------------------+
|       Staking        |      42     | ERC165 |                       |      No      |       Send ETH      |
|                      |             |        |                       |              |     Upgradeable     |
|   IERC20Upgradeable  |      6      | ERC20  |      No Minting       |      No      |     Upgradeable     |
|                      |             |        |  Approve Race Cond.   |              |                     |
| SafeERC20Upgradeable |      6      |        |                       |      No      |       Send ETH      |
|                      |             |        |                       |              | Tokens interaction  |
|                      |             |        |                       |              |     Upgradeable     |
|   AddressUpgradeable |      9      |        |                       |      No      |       Send ETH      |
|                      |             |        |                       |              |       Assembly      |
|                      |             |        |                       |              |     Upgradeable     |
|   StringsUpgradeable |      4      |        |                       |     Yes      |     Upgradeable     |
+----------------------+-------------+--------+-----------------------+--------------+---------------------+
contracts/Staking.sol analyzed (12 contracts)
```

2. Wam

```
Compiled with solc
Number of lines: 533 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 5 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 10
Number of informational issues: 9
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+------+-------------+-------+-----------------------+--------------+----------+
| Name | # functions | ERCS  |       ERC20 info      | Complex code | Features |
+------+-------------+-------+-----------------------+--------------+----------+
| WAM  |      31     | ERC20 |      No Minting       |      No      |          |
|      |             |       |  Approve Race Cond.   |              |          |
+------+-------------+-------+-----------------------+--------------+----------+
contracts/Wam.sol analyzed (5 contracts)
```

Unit Test

```
  Staking contract unit test
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0xdC55E3Df0bd58299eE6ce3DA9a06136B994206c1
    ✓ Should return if staking is allowed
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0x4FC7b57610236898B0203C531Cd5DE42d2fa017a
    ✓ Should be able to stake
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0xbd87AD03336448df3268fa014e4193E250512c4D
BigNumber { value: "297000000000" }
    ✓ Should be able to unstake early (40ms)
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0x3004aFD5d21537b2F76528636A6E5597bD1E99b2
    ✓ Should get correct loses
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0x8155F258887625A097c28C49Df42331Fa8609146
    ✓ Should get correct rewards highest lockup
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0x63de2227739c9420dcC8ec649fBbF7045A42F24d
    ✓ Should get correct rewards medium lockup (48ms)
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0xC54c6b4161958aB6ec539373F58d763d4140E18C
    ✓ Should get correct rewards low lockup
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0xB7535239f9F1f8984EA5F75Df11b120629695B91
    ✓ Should get correct rewards highest lockup - partial unstake
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0xc04f3e9651Dd16E3eA2bc9405398fBb8dB3DE8E6
    ✓ Should get correct rewards highest lockup - partial unstake - then full unstake (46ms)
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0x854cfe7b531095fC9eB57539e6242816EAa23c8f
    ✓ Should get correct rewards partial unstake loss then full unstake proffit (46ms)
```

```
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0x8E489aeF83eeFeb3165522e1005F342e9D8fFcEA
    ✓ Should get correct rewards estimated
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0xA7CcfF2137544EcB42a4c487eAe1ff65254F36aB
    ✓ Should get correct rewards estimated during stake
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0x1b6Be5Ba1D46E8e4A6A92078b9Fd79fbF427F562
    ✓ Should get correct loses estimated during stake
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0xDA511e73a9d806F8E025D684567Ac1a0Ce4C0A7F
    ✓ Should get correct loses estimated during stake - 2
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0x7d07b9b5c72d996754d3940A018551153826FC17
    ✓ Should get correct loses estimated after stake
0x3D17e8069051074DDd10E8BEbaad8fE10Be73B68
Staking deployed to: 0x66B7Feb1da1dc43Dd64d9709ffA03c04ee812433
    ✓ Should update rewardsLeft correctly (41ms)


  16 passing (35s)
```

Concluding Remarks

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

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

Note: The WAM team has refactored the code based on the auditor's recommendation.

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 WAM 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.