PUBLIC REPORT

Dragonereum Smart Contract Security Audit

peppersec

Foreword

Status of a vulnerability or security issue

Clarity is a rare commodity. That is why for the convenience of both the client and the reader, we have introduced a system of marking vulnerabilities and security issues we discover during our security audits.

No issue

Let's start with an ideal case. If an identified security imperfection bears no impact on the security of our client, we mark it with the label.

✓ Fixed

The fixed security issues get the label that informs those reading our public report that the flaws in question should no longer be worried about.

Addressed

In case a client addresses an issue in another way (e.g., by updating the information in the technical papers and specification) we put a nice tag right in front of it.

Acknowledged

If an issue is planned to be addressed in the future, it gets the tag, and a client clearly sees what is yet to be done.

Although the issues marked "Fixed" and "Acknowledged" are no threat, we still list them to provide the most detailed and up-to-date information for the client and the reader.

Severity levels

We also rank the magnitude of the risk a vulnerability or security issue pose. For this purpose, we use 4 "severity levels" namely:

1. Minor
2. Medium
3. Major
4. Critical

More details about the ranking system as well as the description of the severity levels can be found in **Appendix 1. Terminology**.

TABLE OF CONTENTS

16. Contract doesn't prevent zero amount Gold buying — ✓ Fixed

17. Attacker can use rounding to buy Gold for free — ✓ Fixed

18. Passing auction type is redundant — ✓ Fixed

19.Zero item has special meaning — Acknowledged

20. Seller can use front-running attack to change auction and steal GOLD tokens — ✓ Fixed

21.Marketplace doesn't care about accidentally sent ETH — ✓ Fixed

22. GoldMarketplace.totalSupply() returns wrong value — Acknowledged

23. Resending an open or not existing egg to the incubator — Acknowledged

24. pause can't be called directly — ✓ Fixed

25. MainFights doesn't care about accidentally sent ETH — ✓ Fixed

26. Wrong calculation in battle logic — ✓ Fixed

27. User can battle with opponent in sale — ✓ Fixed

28. Bad bet validation — ✓ Fixed

29. MainBase.sendToIncubator returns misleading value — ✓ Fixed

30. The constant MAX_COOLNESS has incorrect value — ✓ Fixed

31. The function _getBaseSkillIndex can be replaced by an array — ✓ Fixed

32. The fucntion upgradeDragonGenes doesn't revert the transaction if not updated genome — ✓ Fixed

33. The user won't be able to buy the special peace skill if the seller has insufficient mana — Acknowledged

34. Expensive upgradable pattern — Acknowledged

35. Buying a skill from the dragon participating in gladiator battle	✓ Fixed
36. Check the compliance of the purchased buff specified in the sale	✓ Fixed
37. Gas optimisation in DragonLeaderboard.update	✓ Fixed
38. Already existing buff isn't taken into account	✓ Fixed
39. fillGoldSellOrder gas optimisation	✓ Fixed
40. Buyer can use front-running attack to change Order price and steal GOLD tokens	✓ Fixed
41. Redundant ETH transfering	Acknowledged
42. Order matching	Acknowledged
43. When using the special peaceful skill for intelligence to itself buff to intellect is reset to zero	✓ Fixed
44. Execution of matchOpponents can exceed block Gas limit	Acknowledged
45. Function is not used	✓ Fixed
46. migrate doesn't validate input addresses	✓ Fixed
47. redundant check for uint16	✓ Fixed
48. Missing input validation in setInternalDependencies	✓ Fixed
49. Users have no idea who is winner before startGladiatorBattle call	✓ Fixed
50. requestRewardForGladiatorBattle is redundant	✓ Fixed
51. Attacker can use EVM feature to set new block for battle	Acknowledged
52. returnBetFromGladiatorBattle has redundant checks	✓ Fixed

53. Attacker can use front-running to constantly get compensation from battle — ✓ Fixed

54. Attacker can use front-running to escape from battle — Acknowledged

55. Function createGladiatorBattle doesn't create an event — ✓ Fixed

56. Functions doesn't check the item existence — ✓ Fixed

57. Solidity version should be locked at final project stage — ✓ Fixed

58. Battle creator can use Sybil attack to get compensation — Acknowledged

1. Severity

01. Introduction

1 Source code

Object	Location
Dragonereum. *Except Gladiator Battle bets functionality*	#2faadda1e23156141b4265c4c8ce878a5c8d9b08

2 Smart contract security audit methodology

A client's information resources may be affected by the following types of threats:

1. Confidentiality violations with resulting information disclosure. A successful violation of the kind leads to information becoming known to people, who should not have access to it: unauthorized personnel, clients, partners, competitors, and third parties on the whole.
2. Integrity violations and consequent modification or corruption of data leading to changes in its structure or content, and to complete or partial destruction of the data.
3. Availability violation (denial-of-service) causes the inability of a user to access data.

The code of a smart contract is automatically and manually scanned for known vulnerabilities and logic errors that can lead to security threats. The conformity of requirements (e.g., White Paper) and practical implementation is reviewed as well. More about the methodology here.

3 Auditors

1. Alexey Pertsev
2. Katerina Belotskaya
3. Roman Storm

02. Summary

Below, you can find a table with all the discovered bugs and security issues listed.

Vulnerability description	Severity
Attacker can use rounding to buy Gold for free	Critical
Seller can use front-running attack to change auction and steal GOLD tokens	
Buyer can use front-running attack to change Order price and steal GOLD tokens	
Weak random for low-value transaction	Major
ERC721Receiver doesn't correspond to ERC721 specification	
Migration script doesn't set required dependency	
pause can't be called directly	
Wrong calculation in battle logic	
Check the compliance of the purchased buff and specified in the sale	
Already existing buff is not taking into account.	
When using the special peaceful skill for intelligence to itself buff to intellect is reset to zero	
Execution of matchOpponents can exceed block Gas limit	
migrate doesn't validate input addresses	
Users have no idea who is winner before startGladiatorBattle call	
Attacker can use EVM feature to set new block for battle	
Attacker can use front-running to constantly get compensation from battle	
Attacker can create fake auctions	Medium
removeGoldFromSale does not clean allowance	
Lack of tx.origin validation	
ERC721Basic	
modifier onlyController is not Gas optimal	

Vulnerability description	Severity
Redundant array copying	Medium
Redundant modifier	
Hardcoded block number	
Missing event	
Old ERC20 Token implementation	
Redundant modifier canTransfer	
Contract does not prevent zero amount Gold buying	
Passing auction type is redundant	
Marketplace does not care about accidentally sent ETH	
GoldMarketplace.totalSupply() returns wrong value	
MainFights does not care about accidentally sent ETH	
User can battle with opponent in sale	
Bad bet validation	
The constant MAX_COOLNESS has incorrect value	
Expensive upgradable pattern	
Buying a skill from the dragon participates in gladiator battle	
Gas optimisation in DragonLeaderboard.update	
fillGoldSellOrder gas optimisation	
Redundant ETH transfering	
Missing input validation in setInternalDependencies	

Vulnerability description	Severity
requestRewardForGladiatorBattle is redundant	Medium
returnBetFromGladiatorBattle has redundant checks	
Attacker can use front-running to escape from battle	
Solidity version should be locked at final project stage	
Battle creator can use Sybil attack to get compensation	
ERC721BasicToken. getApproved doesn't have check that token exists	Minor
Zero item has special meaning	
Resending an open or not existing egg to the incubator.	
MainBase.sendToIncubator returns misleading value	
The function _getBaseSkillIndex can be replaced by an array	
The fucntion upgradeDragonGenes does not revert the transaction if not updated genome	
The user won't be able to buy the special peace skill if the seller has insufficient mana	
Order matching	
Function is not used	
redundant check for uint16	
Function createGladiatorBattle does not create an event	
Functions does not check the item existence	

03. General issues

1 Attacker can create fake auctions

Severity: **MEDIUM**

Marketplace uses the approval mechanism for the `sellGold` and `buyGold` functions. This behavior may be used by an attacker to create a huge amount of fake auctiowns, which may confuse legitimate users.

Recomendations:

1. Consider using actual token transfering instead of approving.

Status:

Acknowledged

Team response:

We are checking user's gold balance on the UI and hiding empty auctions or decreasing amount of selling gold if possible.

2 removeGoldFromSale doesn't clean allowance

Severity: **MEDIUM**

The `removeGoldFromSale` function does not decrease `allowance` during auction removing. In case `Marketplace` is compromised, an attacker can steal all the approved tokens whether a user has an open auction or not.

Recomendations:

1. Consider decreasing `allowance` during `removeGoldFromSale`.

Status:

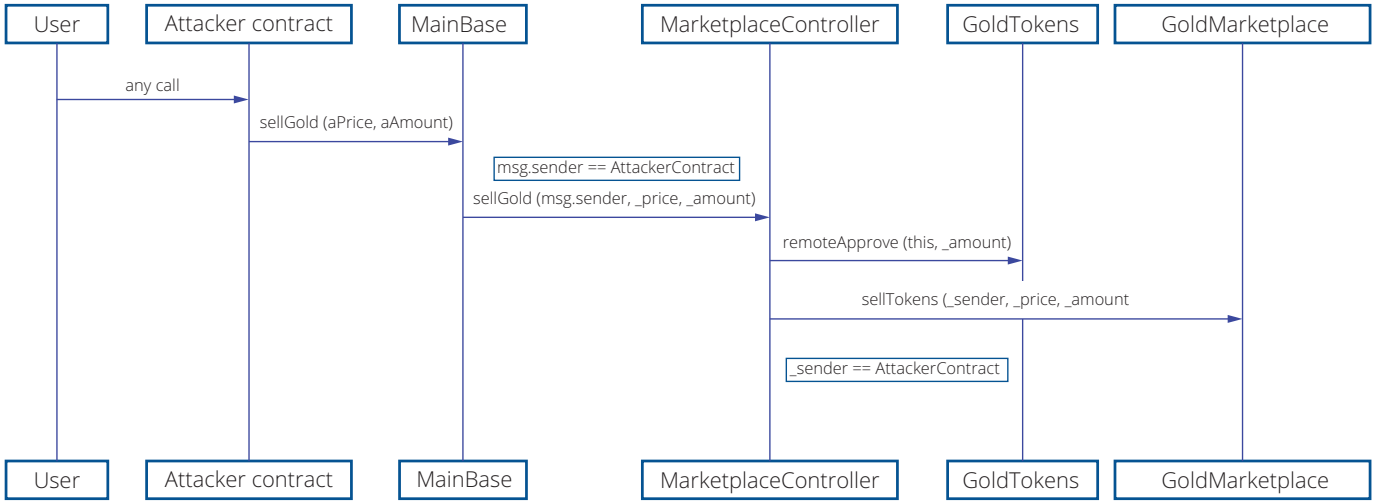
✓ Fixed #0f5649ed7f4151ad6fb3ac8b9637da1fd86e5099

3 Lack of tx.origin validation

Severity: **MEDIUM**

During **sellGold**, Marketplace uses tx.origin to get User address and to approve desirable token amount to Marketplace on User's behalf after that. It also creates auction for selling token. However, in the latter case, Marketplace uses msg.sender.

Attacker can exploit the behavior of the kind to create an auction, which has an attacker's address as the one of a seller, but approved by User. See example below:



Goldtokens.remoteApprove uses tx.origin, which is User.address to approve tokens; while GoldMarketplace.sellTokens uses _sender, which is AttackerContract.address to create an auction.

Recomendations:

1. Consider passing _sender to the remoteApprove function also to avoid possible attacks.

Status:

✓ Fixed d203281140a1c8bbdaa029171fa59e6c5b14f8cd

4 Weak random for low-value transaction

Severity: MAJOR

The game uses the custom approach to getting random numbers for low-value transactions (e.g training battles). According to the white paper and code:

```
rnd = uint256(keccak256(abi.encodePacked(blockhash(block.number - 1), block.timestamp)))
```

> Battle transactions cannot be compromised by a regular player since the RNG includesblock.timestamp is set by a miner. Players can guess what the timestamp will be by looking into historical data and making some assumptions. However, it will still be random as it cannot be determined in advance with certainty.

Indeed, that approach proves to be completely predictable by a player, which means that players can cheat in any low-value transaction:

- the **sendToIncubator** function;
- training battle.

Please, see the example on the next page.

Example:

```
contract Game {
    function getRandom( uint256 _upper) internal view returns (uint256) {
        bytes32 _hash = keccak256(abi.encodePacked(blockhash(block.number), now));
        return uint256(_hash) % _upper;
    }

    function isOdd() external returns(bool win) {
        win = getRandom(2) == 1 ? true: false;
    }
}

contract Exploit {
    Game public target;

    constructor(address _target) {
        target = Game(_target);
    }

    // we just copy necessary part of target contract
    function getRandom( uint256 _upper) internal view returns (uint256) {
        bytes32 _hash = keccak256(abi.encodePacked(blockhash(block.number), now));
        return uint256(_hash) % _upper;
    }

    // cheater calls this func instead of `isOdd`
    function attack() public returns(bool) {
        uint num = getRandom(2);
        if(num == 1) {
            // profit
            return target.isOdd();
        } else {
            revert();
        }
    }
}
```

It works just because all the contracts within the block have the same block.timestamp.

Recommendations:

1. We highly recommend using the same approach as for high-value transactions (future block). If that is not possible in your case then use the following workaround at least:

```
require(msg.sender == tx.origin)
```

Put this requirement to every function that is intended for players and uses random numbers.

Note 1:

That is only a mitigation and works against users. Miners still can decide to put their own tx to block or not to put it and also to manipulate block.timestamp.

Note 2:

The White Paper also should be updated to increase users' awareness.

Status:

✓ Fixed d203281140a1c8bbdaa029171fa59e6c5b14f8cd

5 ERC721Receiver doesn't comply with ERC721 specification

Severity: MAJOR

According to specification, the onERC721Received function must accept 4 arguments:

- _operator – the address that called safeTransferFrom function;
- _from – the address that previously owned the token;
- _tokenId – the NFT identifier that is being transferred;
- _data – additional data with no specified format.

```
function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes _data)
external returns(bytes4);
```

But the function gets 3 arguments without _operator. Also there is incorrectly calculated ERC721_RECEIVED.

Recommendations:

1. Add _operator to the **onERC721Received** function in ERC721Receiver.sol and the **onERC721Received** function to ERC721Holder.sol
2. Fix ERC721_RECEIVED constant in **ERC721Receiver.sol** and in **ERC721BasicToken.sol**. It must be equal to bytes4(keccak256("onERC721Received(address,address,uint256,-bytes)"))
3. Fix the **onERC721Received** function call by passing the first argument to msg.sender: bytes4 retval = ERC721Receiver(_to).onERC721Received(msg.sender, _from, _tokenId, _data);

Status:

✓ Fixed #97053703108d8cac9928a69b8dac00c2437b057c

6 ERC721Basic

Severity: **MEDIUM**

In the **Transfer** event, _tokenId must be indexed.

In the **Approval** event, _tokenId must be indexed.

Recomendations:

1. Consider adding the indexed modifier.

Status:

✓ Fixed #97053703108d8cac9928a69b8dac00c2437b057c

7 ERC721BasicToken. getApproved doesn't have check that token exists

Severity: **MINOR**

The **getApproved** function does not check the existence of a token.

Recomendations:

1. Consider adding **require(exists(tokenId));**

Status:

✓ Fixed #97053703108d8cac9928a69b8dac00c2437b057c

8 modifier onlyController is not Gas optimal

Severity: **MEDIUM**

Every time the onlyController modifier is called, the function goes through the array, referring to the Storage for each array element. According to yellow paper, every SLOAD opcode takes 200 Gas per call.

Recommendations:

1. Consider reorganizing controllers' keeping with the help of mapping:

```
1. contract Controllable is Ownable {
2.     mapping(address => bool) public controllers;
3.     modifier onlyController {
4.         require(isController(msg.sender), "no controller rights");
5.         _;
6.     }
7.
8.     function isController(address _controller) public view returns(bool){
9.         return controllers[_controller];
10.    }
11.
12.    function setControllers(address[] _controllers) internal onlyOwner {
13.        for (uint256 i = 0; i < _controllers.length; i++)
14.        {
15.            require(_controllers[i] != address(0), "invalid address");
16.            controllers[_controllers[i]] = true;
17.        }
18.    }
19.}
```

Status:

✓ Fixed #b25a88c8057c4d33fb7e576e6af8476680d74cc9

9 Redundant array copying

Severity: **MEDIUM**

The `upgradeDragonGenes` function copies `_points` array to Memory array before calling `coreController.upgradeDragonGenes` function. This action is not necessary but takes additional Gas.

Recommendations:

1. Consider removing **line 58**.

Status:

✓ Fixed **#ae6435d6724b664a5edd61488753beba05430ab7**

10 Redundant modifier

Severity: **MEDIUM**

The `onlyController` modifier is used for many `views`, which is redundant and can be harmful for the frontend.

Occasions:

1. **EggCore.sol#L9**
2. **EggCore.sol#L13**
3. **EggCore.sol#L17**
4. **EggCore.sol#L21**
5. **EggCore.sol#L37**

Recommendations:

1. Just remove it.

Status:

✓ Fixed **#a5bc036362b49554120e785bf1b1046b8b3c8272**

11 Migration script doesn't set required dependency

Severity: MAJOR

The `3_establish_dependencies.js` migration script does not set the `Events` contract as a dependency of Mainbase.

Recommendations:

1. Add the `Events` contract to the dependency list.

Status:

✓ Fixed f00f9e9ef4522aae698efd048275ca6eecbb8bbd

12 Hardcoded block number

Severity: MEDIUM

The `Distribution` contract utilizes hardcoded block number for work.

Recommendations:

1. Consider using `block.number` to set `lastBlock` in the constructor to avoid issues in different chains.

Status:

✓ Fixed f00f9e9ef4522aae698efd048275ca6eecbb8bbd

13 Missing event

Severity: MEDIUM

The `setName` function does not emit any events.

Recommendations:

1. Consider implementing event firing. It would be useful for frontend part of the DApp.

Status:

✓ Fixed ee6d4ae86a5225f46c4bef4530b30855d67fb00d

14 Old ERC20 Token implementation

Severity: MEDIUM

The current implementation of the ERC20 token is deprecated and vulnerable to withdrawal attacks.

Recommendations:

1. Consider updating to the latest version of the OpenZeppelin implementation: add the increaseAllowance and decreaseAllowance functions.

Status:

✓ Fixed 819a33214fccd4b3788255f092660c5ee0c0f343

15 Redundant modifier canTransfer

Severity: MEDIUM

Both canTransfer(_tokenId) in the safeTransferFrom(address _from, address _to, uint256 _tokenId) function and canTransfer(_tokenId) in the safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes _data) function are redundant due to this.

Recommendations:

1. Consider removing the usage of that modifier.

Status:

✓ Fixed 81bd0a33e7519d25a27cc79b86b0b988049e643c

16 Contract doesn't prevent zero amount Gold buying

Severity: MEDIUM

The buyGold(address _seller, uint256 _amount) function does not require(_amount !=0). So a user can be frustrated by the behavior of the kind.

Recommendations:

1. Consider adding a check that require(_amount !=0).

Status:

✓ Fixed #d271ad0475d61e515b64762f5781f02bf1c9ebea

17 Attacker can use rounding to buy Gold for free

Severity: **CRITICAL**

The buyTokens function expects the _amount and price as values like N * 10^{18}, which is not always true. So if the price of a token equals 0.00001 * 10^{18} then the expression:

```
fullPrice = auction.price.mul(_amount).div(10**18);
```

gives 0 if _amount is less than 99999 * 10^{18}. This means that an attacker can get free GOLD tokens.

Note:

An attacker should always pay a commision for a transaction, that is why those tokens are not free and non-profitable either. However, if the price increases over time then those “stolen tokens” may become profitable.

Recommendations:

1. Consider adding a check that fullPrice > 0 after calculation.

Status:

✓ Fixed #6ed8b75b34ba7865710379113d34831afde80082

18 Passing auction type is redundant

Severity: **MEDIUM**

The sellEgg, sellDragon, and sellBreeding functions require the _auctionType argument. After that, sellToken carefully checks that it is right before creating an actual auction.

Recommendations:

1. Consider removing the _auctionType arg and implementing auction type detection based on _startPrice and _endPrice.

Status:

✓ Fixed 3335fe98bf03f05834c7d0aa307a3724bb879ee2

19 Zero item has special meaning

Severity: **MINOR**

Arrays of Eggs, Dragons, and Battles have a special zero item to avoid some issues with 0. For instance, “some” issue for egg is that every egg has parents that are just numbers in same Eggs array. By design, genesis eggs have no parent, so such eggs hold the [0,0] array as parents, thereby this zero item is a parent for all genesis eggs. That is why the constructor sets eggs.length to 1.

Recommendations:

1. To avoid this kind of issues, one can use magic numbers. For example, you can use 2**256-1 as a parent for genesis eggs.

Status:

Acknowledged

Team’s response:

We would not like to change this logic if there are no possible vulnerabilities in it.

20 Seller can use front-running attack to change auction and steal GOLD tokens

Severity: **CRITICAL**

By design, sellers can change their existing auctions at any time by calling the sellEgg, sellDragon and sellBreeding functions second time. On the other hand, users pick only the ID of the items they can buy. That gives an attacker the opportunity to steal user coins by changing the auction *after* users send their transaction (with the id of the items they can buy) and before the smart contract processes it.

Example:

1. Attackers set a cheap auction for an incredibly powerful egg (let’s say, the auction is in GOLD tokens).
2. A user sees that and tries to buy it.
3. When attacker monitor pending pool and when the tx appears there, they sends their tx with a higher GasPrice to reset auction.
4. When the user’s tx reaches the smart contract, supplied _id will point to the auction with a price that equates to the whole GOLD token balance of the user (which can be way more than the user wanted to spend).

The same technique can be used to change an ETH auction to a GOLD auction with a higher price in the token equivalent.

Recommendations:

1. Consider adding the `_price` and `_isGold` arguments to the `buy(Egg|Dragon|Breeding)` function to eliminate an attack.

Status:

✓ Fixed #f5e8615226c848d57e704e9b1cbe2b02492c66d8

✓ Fixed #2966ffd2d0e0db93a914c706daeb3d263778628e

21 Marketplace doesn't care about accidentally sent ETH

Severity: **MEDIUM**

The `buyEgg`, `buyDragon`, and `buyBreeding` functions do not check whether a user sends ETH to GOLD auction or not. So, in case of a bug on frontend or something like this, the sent ETH will be locked forever.

Recommendations:

1. Consider adding a check like the following one:

```
if (_isGold) {
    require(msg.value == 0, “this is GOLD auction”);
}
```

Status:

✓ Fixed #ed76c53af09bad518bcb7a8350cd3b943b483309

22 GoldMarketplace.totalSupply() returns wrong value

Severity: **MEDIUM**

The `GoldMarketplace.totalSupply()` function returns the current count of open auctions by returning the `length` of `auctions` . Nonetheless, due to the **Zero item has special meaning** issue, it is one item more than actually exists.

Recommendations:

1. Consider decreasing value or resolve the **Zero item has special meaning** issues.

Status:

Acknowledged

Team's response:

We skip zero item on UI. If someone get zero item manually then he will not be able to do anything with it.

23 Resending an open or not existing egg to the incubator

Severity: **MINOR**

At the time of **sending an egg to the incubator**, there is no explicit verification that the egg actually exists. The transaction is interrupted by the message **"invalid address"**.

Recommendations:

1. Consider adding a check that an egg exists and has not been previously opened.

Status:

Acknowledged

Team's response:

Then we need to forward exists function through the EggCore and Core contracts. I can't see an issue with the exception of a non-obvious error text

24 pause can't be called directly

Severity: **MAJOR**

Due to the current architecture, it is not possible to call `pause` of a particular contract directly. Since `UpgradeController` is the owner of each contract, before calling `pause`, the team has to `returnOwnership` of a particular constract and then call `pause`. The action has to be repeated for each single `Pausable` contract. Then, we need to call `transferOwnership` for each contract to return the ability to `migrate` contracts to `UpgradeController`.

Recommendations:

1. Consider implementing the function contract that can pause all contract per one transaction into UpgradeController.

Status:

✓ Fixed #b00037eb96ba5eb51278f2c62565ab5b30bc91f7

25 MainFights doesn't care about accidentally sent ETH

Severity: **MEDIUM**

The `createBattle` and `applyForBattle` functions do not check whether a user sends ETH when `isGold == true` or not. So, in case there is a bug on frontend or something like this - sent ETH will be locked forever.

Recommendations:

1. Consider adding a check like the following one:

```
1. if (_isGold) {
2.     require(msg.value == 0, "specify isGold as false to send ETH");
3. }
```

Status:

✓ Fixed #4ded57b3c7bc8772702778ac16e3f2b4200d4e38

26 Wrong calculation in battle logic

Severity: **MAJOR**

The line 318 of the `_turn` function enables a dragon to make a step back if it is too close for a long-range attack. However, instead of changing its current `distance`, it sets the distance to `0.7 * DragonSpeed` (it is constant for particular dragon). It can lead to bugs in the game mechanics.

Recommendations:

1. Consider removing the comparison with MAX_DISTANCE (it is not achievable) and changing the “step back” logic.

Status:

✓ Fixed #cabc85746eaf9bbbb6c02c0da330d6742c0bf136

27 User can battle with opponent in sale

Severity: **MEDIUM**

`_checkBattlePossibility` requires that an owned dragon is not on sale (or breeding), but not requires the same from the opponent.

Recommendations:

1. Consider adding the check for the opponent.

Status:

✓ Fixed #9025d1df3d894bc70a29aa8438a9123380cefc32

28 Bad bet validation

Severity: **MEDIUM**

The `_payForBet` function requires `msg.value >= _bet`, but the fact that `msg.value` is more than the bet means that something is wrong.

Recommendations:

1. Consider adding the sign to `==`.

Status:

✓ Fixed #4ded57b3c7bc8772702778ac16e3f2b4200d4e38

29 MainBase.sendToIncubator returns misleading value

Severity: **MINOR**

The `sendToIncubator` function returns the `newDragonId` value after an incubated egg, but a new dragon does not match the egg that has been sent by the user to the incubator in this transaction and the user gets an ID of someone else's dragon.

Recommendations:

1. Consider removing `newDragonId` from return values.

Status:

✓ Fixed #4692dff19f41e30744f3008b687e95e6396b4303

30 The constant MAX_COOLNESS has incorrect value

Severity: **MEDIUM**

The constant `MAX_COOLNESS` is equal to `6336` (= 1.6 * 99 * 10 * 4), if max `GeneVarietyFactor` is equal to `1.6`, but in fact the maximum is `2.8`.

Recommendations:

1. Consider using the value `11088` (= 2.8 * 99 * 10 * 4).

Status:

✓ Fixed #4ab038ec3bc93763b60ef859184a0b81ccde4b4d

31 The function _getBaseSkillIndex can be replaced by an array

Severity: **MINOR**

The `_getBaseSkillIndex` function can be replaced by an array to improve code readability.

Recommendations:

1. Consider using the array.

Status:

✓ Fixed #bde2f6e4fd09b4938b5651bba297646ccee836be

32 The fucntion upgradeDragonGenes doesn't revert the transaction if not updated genome

Severity: **MINOR**

If the `upgradeDragonGenes` function takes the `_dnaPoints` array (for example: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]), and the sum of the elements is not greater than the available `dnaPoints` (for example: 10), but the values of each element are not enough to update the genome, the genome will not be changed and the fucntion will not revert the transaction.

Recommendations

1. In the `upgradeGenes` function, use the flag that is set if the genome is changed, and check this flag. Another option is to compare the new genome and the old one in the `upgradeGenes` function. In any case, if the check fails, reject the transaction.

Status:

✓ Fixed #dfa6fd6356b10d9106413b2404078edd52a308c2

33 The user won't be able to buy the special peace skill if the seller has insufficient mana

Severity: **MINOR**

Status:

Acknowledged

Team's response:

On the web site we hide such auctions from users.

34 Expensive upgradable pattern

Severity: **MEDIUM**

The current architecture consists of many contracts that refer to each other by storing dependency addresses. This scheme forms a multilayer structure, where there are several entry points (contracts like `mainBase`, `mainBattle`, and so on). Each call to them is passed to lower level, which implements the actual logic.

1. Each call of external smart contract consumes 2300 Gas. Some functions have 4 or more external calls in the process of execution, which consumes additional Gas.
2. Each `internalDependency` must store all possible `externalDependencies` and strictly check `msg.sender` for each call, which consumes additional Gas.
3. In case of any changes made to the smart contract that stores user data (`DragonStorage`, `Gold`, and so on), that data should be moved (expensive operation).
4. In case of entry point changing, the user interface should change that smart contract address.

Team's response:

Due to huge size of smart contracts we cannot use modern Upgradability mechanisms (like Zeppelin ones), but we plan fix it future releases.

Status:

Acknowledged

35 Buying a skill from the dragon participating in gladiator battle

Severity: **MEDIUM**

If a dragon's skill is exhibited on sale and it participates in the gladiator battle, another user can purchase the skill, thereby removing mana and buffs.

Recommendations:

1. Consider prohibiting skill sales if a dragon participates in the gladiator battle.

Status:

✓ Fixed #5410338e979038cd18b4f19951a58ebaed780532

36 Check the compliance of the purchased buff specified in the sale

Severity: **MAJOR**

When buying a special peace skill, it is not checked that the effect of the bought buff corresponds to the one specified at the sale.

Recommendations:

1. Consider adding a parameter, value buff, expected by the buyer in the buySkill function.

Status:

✓ Fixed #978360ec77639b0291455d27524a5b7f29ec33fc

37 Gas optimisation in DragonLeaderboard.update

Severity: **MEDIUM**

Recommendations:

1. Consider adding the following:

```
1. if(_isIndex && _isExistingIndex) break;
```

to line 44 to avoid redundant (expensive) Storage accesses.

✓ Fixed #49a40318063a3fe7c70063a9202bdf804547aa27

38 Already existing buff isn't taken into account

Severity: **MAJOR**

If a buffed dragon uses a special peace skill the calculateSpecialPeacefulSkill function does not use this buff when calculating the effect and cost of the skill.

Recommendations:

1. Take into account the already existing buff in the process of special peace skill counting.

Status:

✓ Fixed #bff4b28036694d860c7705bfb76f7a2bcc5b4ff7

39 fillGoldSellOrder gas optimisation

Severity: **MEDIUM**

The `fillSellOrder` function updates `SellOrder` even if the latter is empty and only then removes it.

Note:

The `fillBuyOrder` acts the same way.

Recommendations:

```
1. _available = _available.sub(_amount);
2.
3. if (_available == 0) {
4.     _storage_.cancelSellOrder(_seller);
5. } else {
6.     _storage_.updateSellOrder(_seller, price, _available);
7. }
```

Status:

✓ Fixed #37b2fcb729646e2e0e2a6d781c1def11ecd76c3a

40 Buyer can use front-running attack to change Order price and steal GOLD tokens

Severity: **CRITICAL**

The game allows anyone to create a “buy” Order by calling `createGoldBuyOrder(price, amount)`. Also, buyers can `cancel` their “buy” orders at any moment.

On the other hand, users pick only a buyer’s address and the amount (not the price) to “fill” their Order.

That provides an attacker with the opportunity to chage the actual “price” and a steal user’s coins by canceling the current Order and creating a new one *after* the user sends the transaction and *before* the smart contract processes it.

Note:

The same technique can be used to change a “sell” Order and steal ETH if a user sends the surplus.

Recommendations:

1. Consider adding the `price` argument to the `fillGoldBuyOrder` and `fillGoldSellOrder` functions to eliminate the attack.

✓ Fixed #ce30cdb1a4a6953c2313fd9cee34111279bb800c

41 Redundant ETH transfering

Severity: **MEDIUM**

The **createGoldBuyOrder** function transfers a user’s ETH to the `GoldMarketplace` contract, which, in its turn, transfers it to the `GoldMarketplaceStorage` (which actually keeps it).

Recommendations:

1. Consider transferring ETH to `GoldMarketplaceStorage` directly.

Status:

Acknowledged

Team’s response:

We transfer eth to GoldMarketplace in `fillSellOrder` so we want to keep the same logic in `createGoldBuyOrder`.

42 Order matching

Severity: **MINOR**

Dragonereum is capable of trading GOLD token. There are functions like `createGoldSellOrder`, `createGoldBuyOrder`, `fillGoldBuyOrder`, and others. However, the current implementation does not match a user's "buy" and "sell" orders in cases when it is possible.

Recommendations:

1. Consider implementing the functionality into the front-end part of the game.

Acknowledged

Team's response:

We are going to implement that in the next release.

43 When using the special peaceful skill for intelligence to itself buff to intellect is reset to zero

Severity: **MAJOR**

When players use their `special peaceful skill` for intelligence on themselves **their buff is reset to zero**.

Recommendations:

1. Consider removing `reset intelligence buff of the first dragon` for this case.

Status:

✓ Fixed **#bff4b28036694d860c7705bfb76f7a2bcc5b4ff7**

44 Execution of matchOpponents can exceed block Gas limit

Severity: **MAJOR**

The matchOpponents function can exceed block Gas limit in case of existing too many dragons.

Recommendations:

1. Consider adding off-chain matching.

Status:

Acknowledged

Team's response:

We are going to implement opponent matching on a backend in next release.

45 Function isn't used

Severity: **MINOR**

The internal function _getHash is not used anywhere.

Recommendations:

1. Consider removing the function.

Status:

✓ Fixed #77c2e21f1d6af1ad5030fa1b2cd087431002cc2e

46 migrate doesn't validate input addresses

Severity: MAJOR

The migrate function does not validate _oldAddress and _newAddress at all.

Recommendations:

1. Consider adding the following validation:
   - _oldAddress != _newAddress
   - "Old" contract should have external or internal dependencies

Status:

✓ Fixed #33518006f41e0556ff1aaa32e120fd8332f2b06f

47 redundant check for uint16

Severity: MINOR

The create function requires the _counter < 65536. However, it is not neccessary because _counter has uint16 type, so it is less than 65536 by default.

Recommendation:

1. Consider removing the check.

Status:

✓ Fixed 78554b58ec77b9c76c38de91a574d9193dbe1004

48 Missing input validation in setInternalDependencies

Severity: MEDIUM

The setInternalDependencies function does not validate dependencies' addresses.

Recommendation:

1. Consider adding validation.

Status:

✓ Fixed #d5d624b6403d84a42965825fbf65cf8ee69c2d9f

49 Users have no idea who is winner before startGladiatorBattle call

Severity: **MAJOR**

The `startGladiatorBattle` function allows anyone to draw the battle. However, the current user interface does not enable users to make sure they are have won, which is technically possible. The smart contract can have the special `view` function, which returns a winner of a particular battle if it is possible. Once users can look at battle results, losers will save their funds since only an actual winner calls `startGladiatorBattle`.

Recommendations:

1. Consider improving the code in the described way.

Status:

✓ Fixed #3295de452e1910ab87df67cb1b0a8d141e919745

50 `requestRewardForGladiatorBattle` is redundant

Severity: **MEDIUM**

The `requestRewardForGladiatorBattle` function allows the winner to get the reward. Indeed, the winner can be rewarded by the `startGladiatorBattle` function, it saves users' ETH and speeds up battle process.

Recommendations:

1. Consider improving the code in the described way.

Status:

✓ Fixed #7002f382670b0e25b808d912cd0ab0373798d7c2

51 Attacker can use EVM feature to set new block for battle

Severity: **MAJOR**

Accoding to the Solidity documentation:

The block hashes are not available for all blocks for scalability reasons. You can only access the hashes of the most recent 256 blocks, all other values will be zero.

Due to this fact, the contract **validates** that properly. However, the case when no one calls **startGladiatorBattle** is still the thing.

To avoid funds locking, the contract has the **updateBlockNumberOfGladiatorBattle** function that allows moving the block to future.

Attack scenario:

1. An attacker creates a battle with a long “free to apply” period.
2. Some users apply for the battle, but they cannot be online all the time (let’s imagine they go to eat and sleep sometimes).
3. An attacker waits for an appropriate time to call chooseOpponentForGladiatorBattle with one on users’ dragon (let's suppose, it happens at 3 AM in the victim's timezone)
4. Now, if an attacker loses (it is possible to investigate that without an actuall call of startGladiatorBattle). After that, an attacker just needs to wait for 256 blocks (**about an hour**) to call updateBlockNumberOfGladiatorBattle and reset the battleBlockNumber.
5. After that, an attacker gets another chance to win.

Recommendations:

1. You can create a watcher, which keeps track of battles and calls startGladiatorBattle if a user is offline.

Status:

Acknowledged

Increased the number of added blocks:
#c6ed08cd514abd4d747eda27371d143e98dda411

Team’s response:

This is a global problem for Ethereum. The user can control this situation himself so as not to lose money. A informing message the user about this will be added also.

52 returnBetFromGladiatorBattle has redundant checks

Severity: **MEDIUM**

The `returnBetFromGladiatorBattle` function checks that **caller is not one of participants** (the creator of an opponent) which is redundant because of previous check that user is **one of applicants**.

Recommendations:

1. Consider removing redundant checks to save Gas.

Status:

✓ Fixed 4db6af19f6d91200df88c9dc308be1ff34099fb6

53 Attacker can use front-running to constantly get compensation from battle

Severity: **MAJOR**

An attacker can use a front-running attack to add a lot of other dragons (calling `applyForGladiatorBattle`) to battle right before the `creator` calls `chooseOpponentForGladiatorBattle`. Since all rest users (non-selected) give a `bet` + `compensation`, the attacker can get stable profit by participating in battles in that way.

The same issue is with `cancelGladiatorBattle`: attackers can monitor all battles and put their dragons to battle right before canceling.

P.S. The creator will not see an attacker's dragons just because the user interface cannot show them. The interface shows only things that already happened (mined).

Recommendations:

1. You can add an additional function for battle creator that will freeze `applyForGladiatorBattle` for some time.
2. Or add the opponents array list as an argument for `chooseOpponentForGladiatorBattle` and `cancelGladiatorBattle`. In this case contract have to check equality of that list and **challengeApplicants**.

Note:

An attacker can also use `returnBet` to escape from a battle. See the **Attacker can use front-running to escape from battle** issue.

Status:

✓ Fixed **#1f9373ca5daab13f21d62d83ad8e94ba184e7d57**

✓ Fixed **#e9af2fe110698c4207d36979a53d65c6049fad5e**

54 Attacker can use front-running to escape from battle

Severity: **MEDIUM**

An attacker can use a front-running attack to escape (calling `returnBetFromGladiatorBattle`) from the battle right before the creator calls `chooseOpponentForGladiatorBattle`. In this case, the creator's transaction fails and a user could be confused.

Recommendations:

1. The creator should have an opportunity to hide the choice if there are too many opponents. Consider using the commit-reveal scheme to accomplish that.

Status:

Acknowledged

Team's response:

Too complicated solution for this case. The only thing that can happen is that the user's transaction will fail and he will be confused

55 Function createGladiatorBattle doesn't create an event

Severity: **MINOR**

Missing event emition in the createGladiatorBattle function.

Recommendations:

1. Consider adding the event of the successful creation of the battle.

Status:

✓ Fixed #b057b15ff266a071ba02e19c1323f335d0b765c3

56 Functions doesn't check the item existence

Severity: **MINOR**

Functions from Getter.sol and DragonGetter.sol does not check the existence of the dragon and returns zero values for a non-existent dragons.

Recommendations:

1. Consider adding the checks.

Status:

✓ Fixed #29f53f255390ca13c99ee7d1bd5194b3806be6c5

57 Solidity version should be locked at final project stage

Severity: **MEDIUM**

Contracts should be deployed with the same compiler version and flags that they have been tested most with. Locking the pragma, helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler, which may pose higher risks of undiscovered bugs. Contracts may also be deployed by others and the pragma indicates the compiler version intended by the original authors.

Recommendations:

1. Consider locking on a specific pragma version.

Status:

✓ Fixed #6eb96bc5c2e7522134c1d8b1fd40c47ff1e8f416

58 Battle creator can use Sybil attack to get compensation

Severity: **MEDIUM**

A battle Creators can `applyForGladiatorBattle` their dragons against themselves. Despite the function checks the existing `userApplication`, it still can be accomplish by controling two or more Ethereum addresses that have dragons (read more about the **Sybil attack**). So, the battle creator may have up to 30% of the bet as a `compensation` regardless of the outcome of the battle.

Recommendations:

1. That is an acrhitectual feature of decentralized systems. An attack cannot be eleminated without serious protocol changes.

Status:

Acknowledged

Team's response:

We plan gladiator battle protocol enhancing in the next release.

05. White Paper compliance

This section describes the differences between the ideas described in the WP and the actuall smart-contract implementation (all fixed).

1. page 7

"Every dragon has its unique, immutable set of genes"

Comment: a player can upgrade genes.

2. page 10.

"... if the dragons have an equal amount of HP left, the battle result is a draw and both dragons are considered as defeated and norewards are allocated."

Comment: **according to the code**, the first player will win.

3. page 10.

"The game's creators will not have any control over Gold stored in the Dragonereum Gold account once it has been deployed tothe blockchain."

Comment: upgradable mechanism allows everything.

4. page 10.

"The value of the reward will be aligned with the number of dragons in the population and the number of battles held..."

Comment: the reward depends on "coolness", dragons in the population and rest of Gold, but does not "number of battles held".

5. page 11.

"Now other players (every player not necessarily those who participate in this fight)can place bets..."

Comment: is not implemented.

6. page 15.

"Dragonereum's solution" or "Low-value transactions" section.

Comment: as we described it in the **"Weak-random-for-low-value-transaction"** issue, it is easily predictable when last block hash is used. Consider describing that case and your countermeasures (only human modifier).

7. page 45.

The business logic layer consists of several smart contracts: egg core features contract,dragon core features contract, breeding, battling, marketplace and auth contract.

Comment: which one is "auth"?

8. page 46.

However, to make the life of players a bit easier another back-end server is used for:

- Filtering of items offered on marketplaces
- Storage and delivery of notifications while a player is not online.

Recommendation: consider adding "battle opponent choosing" and "gladiator battle notificaton".

9. page 19, 20.

```
struct Fights {
      uint16 wins;
      uint16 defeats;
}
 mapping (uint256 => Fights) public fights;
```

Recommendation: consider renaming Fights **to** Battles**. 10. page 19**

10. page 19.

```
struct Dragon {
      uint16 generation;
      uint256[4] genome;
      uint256[2] parents;
      uint8[11] race; // array of weights of dragon's types
      uint256 birth; // timestamp
}
```

Recommendation: consider renaming `race` to `types`.

11. Upgradable mechanism is not mentioned

Recommendation: consider describing the implemented Upgradable mechanism.

12. Page 51.

"The multisig address ofNonsense Games is 0x2a52fba0c290d30214cb7876ede81c-2b407e8782"

Comment: Are you sure about the address? Etherscan says **there is no mutisig there**

13. Page 20.

```
mapping (uint256 => uint32) public DSI; // Dragon Skillfulness Index multiplied by 100
```

Recommendation: consider renaming `DSI` to `coolness`.

14. Order book is not mentioned.

Recommendation: consider describing the GOLD trade market feature.

Conclusions

During the audit there were 3 **Critical** and 14 **Major** bugs discovered. By exploiting them, an attacker could:

- steal funds from legitmate users (like GOLD tokens and ETH);
- predict and exploit all “low-value” functions (regular battle, breeding, egg nesting, etc.);
- launch front-running attacks to cheat in Gladiator Battles and buy GOLD tokens with lower price in Order Book.

Also, the auditors investigated a wide range of security weaknesses and offered a lot of ways for Gas optimization, which would certainly help to reduse a transaction cost.

As it can be seen above, all the specified issues have been resolved or the client's team has acknowleged them. The client's prompt response to the discovered vulnerabilities and security issues has significantly increased the overall security level of the platform.

Appendix 1. Terminology

1 Severity

Severity is the category that described the magnitude of an issue.

		Severity		
Impact	Major	Medium	Major	Critical
	Medium	Minor	Medium	Major
	Minor	None	Minor	Medium
		Minor	Medium	Major
		Likelihood		

MINOR

Minor issues are generally subjective in their nature or potentially associated with the topics like “best practices” or “readability”. As a rule, minor issues do not indicate an actual problem or bug in the code. The maintainers should use their own judgment as to whether addressing these issues will improve the codebase.

MEDIUM

Medium issues are generally objective in their nature but do not represent any actual bugs or security problems. These issues should be addressed unless there is an apparent reason not to.

MAJOR

Major issues are things like bugs or vulnerabilities. These issues may be unexploitable directly or may require a certain condition to arise to be exploited. If unaddressed, these issues are likely to cause problems with the operation of the contract or lead to situations which make the system exploitable.

CRITICAL

Critical issues are directly exploitable bugs or security vulnerabilities. If unaddressed, these issues are likely or guaranteed to cause major problems and ultimately a full failure in the operations of the contract.

Appendix 2. Battle protocol

The protocol was compiled, in accordance with the changes made after the security audit.

1. User call `createGladiatorBattle(dragonId, tactics, isGold, bet, counter)`, where:

- dragonId - id of the owned dragon
- tactics - the tactics for the dragon
- isGold - bet made in Gold or not
- bet - if the bet is made in Gold, that its amount
- counter - a counter that defines “the number of future blocks after which the battle will be initiated”
- if isGold == false, than ETH equal to bet is required to be sent. After that, the battle will appear in the list of available battles, and other users will be able to join it (see stage 2).

2. Anyone who feels like it can call `applyForGladiatorBattle(battleId, dragonId, tactics)`, where:

- battleId - id of the battle created at stage 1.
- dragonId - id of the dragon sent to the battle.
- tactics - the tactics for the dragon.
- if the battle's isGold == false, than extra ETH equal to that in stage 1 must be sent. If the battle is held for GOLD, the amount of GOLD will be automatically charged off a user's address.

3. After at least one opponet joins the battle, the creator can call `chooseOpponentForGladiatorBattle(battleId, opponentId, _applicantsHash)`, where:

- battleId - id of the battle created at stage 1
- opponentId - id of the battled dragond (from the list of those who called `applyForGladiatorBattle`)
- applicantsHash - array hash of those who applied for the battle `applyForGladiatorBattle`)

4. After the block locked at stage 3 is sent to past blocks, anyone will be able to call `startGladiatorBattle(battleId)`, where:

- battleId - id of the battle created at stage 1

The winner gets x2 bet (see stage 1) if there has been only 1 opponent, or x1.7 bet if there have been more opponent apart from the chosen contestant (0.3 bet is kept back as commission).

5. Those who have applied for the battle but have not been chosen for it, can get their bet back by calling `returnBetFromGladiatorBattle(battleId)`, where:

- battleId - id of the battle created at stage 1

They get the comp in the amount of their bet + 0.3 bet comp (see stage 4). The whole amount is split equally among the applicants who have not been chosen for the battle.

6. If the battle creator hold off with the choise of the contestant.

After a certain amount of time expires (6k blocks by default) anyone will be able to call `autoSelectOpponentForGladiatorBattle(battleId, applicantsHash)`, where:

- battleId - id of the battle created at stage 1
- applicantsHash - array hash of those who applied for the battle (by calling `applyForGladiatorBattle`)

The contract appoints the contestant itself. To define the contestant's number, it uses the hash of a precedent block.

7. The creator also can cancel the battle if no one is still chosen (see stage 3). That is done by calling `cancelGladiatorBattle(battleId)`. If there are any applicants for the battle, they are give the compensation described in stage 5.

8. The creator can add extra time for `autoSelectOpponentForGladiatorBattle` to be called by calling `addTimeForOpponentSelectForGladiatorBattle(battleId)`. That will add 6000 blocks and will cost additiona 50 GOLD for the first instance, 100 - for the second, 200 - for the third, etc.

9. If no one called startGladiatorBattle during the execution of 256 blocks, anyone would be able to call updateBattleBlockNumber and update battle block. The latter would become the main block + 1000 (approximately 4x hours), as specified in stage 1.

Appendix 3. GOLD transfer protocol

Selling

1. A user can put its GOLD on sale with the help of `createGoldSellOrder(price, amount)`, where:

- price - actual selling price
- amount - amount of GOLD

2. A purchaser can buy the GOLD with `fillGoldSellOrder(seller, amount, price)`, where:

- seller - the user from stage 1
- amount - the amount of GOLD required by the seller
- price - same amount as that in stage 1

3. First user can cancel the order at any given moment by calling `cancelGoldSellOrder()`.

Purchasing

1. If there is no fair selling offer, a user can create a purchase order by calling `createGoldBuyOrder(price, amount)`, where:

- price - the offered price
- amount

2. The seller can sell the GOLD to the purchaser with the help of `fillGoldBuyOrder(buyer, amount)`, where:

- buyer - the user from stage 4
- amount - the amount of the offered GOLD
- price - same amount as that in stage 1

3. The order owner can cancel it at any given moment by calling `cancelGoldBuyOrder()`.

About Us

Worried about the security of your project? You're on the right way! The second step is to find a team of seasoned cybersecurity experts who will make it impenetrable. And you've just come to the right place.

PepperSec is a group of whitehat hackers seasoned by many-year experience and have a deep understanding of the modern Internet technologies. We're ready to battle for the security of your project.

LET'S KEEP IN TOUCH


peppersec.com


hello@peppersec.com


Github