verichains

SECURITY AUDIT OF

DARKLAND SMART CONTRACTS

Public Report

Mar 03, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Polygon	Polygon is a protocol and a framework for building and connecting Ethereum-compatible blockchain networks. Aggregating scalable solutions on Ethereum supporting a multi-chain Ethereum ecosystem.
MATIC	A cryptocurrency whose blockchain is generated by the Polygon platform. Matic is used for payment of transactions and computing services in the Polygon network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Mar 03, 2022. We would like to thank the Darkland for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.This audit focused on identifying security flaws in code and the design of the Darkland Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some issues in the application. Darkland team has resolved and updated all the recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Darkland Smart Contracts

Dark Land Survival is more than just an NFT IDLE Zombie Defense Game powered by blockchain technology. Built on BSC, it's a massive open world that comes with a whole new perspective. Aside from Play-to-Earn mechanic with a rich story, Dark Land Survival offers a superior gaming experience with various gameplay modes and features such as campaign, dungeon, raid mode, construction mode, landlord, and more. All of which encourage players to keep playing and exploring.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of Darkland Smart Contracts. It was conducted on the source code provided by the Darkland team.

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The following files were made available in the course of the review:

FILE	SHA256 SUM
Token.sol	bdff6df220526f0233608453e2d56b53544d5f447e447971e14db68094e11a59
Farms.sol	63d738f4c7bd0ec9c62cb527ca9425bc6bb991cb9ed4eddeedae1cd088c63c99
Pools.sol	f3fc30f3b08e678c24986003146de6aa602c4bf81470f51a8cf7d8a656a19f4f

2.2. Contract code

The Darkland Smart Contracts was written in Solidity language, with the required version to be ^0.8.2. The source code was written based on OpenZeppelin's library.

The provided source codes consist of three contracts which inherit some contracts from OpenZeppelin.

2.2.1. Bigcat token contract

The Bigcat token is an ERC20 token contract. It extends ERC20, Pausable, ERC20Snapshot and Ownable contracts. With Ownable by default, Token Owner is contract deployer, but he can transfer ownership to another address at any time. He can pause/unpause contract using Pausable contract, users can only transfer tokens when contract is not paused. ERC20Snapshot help Token Owner takes a snapshot of the balances and total supply at a time for later access.

Table 2 lists some properties of the audited DeHR token contract (as of the report writing time).

PROPERTY	VALUE
Name	Big Cat Token
Symbol	BIG
Decimals	18

PROPERTY	VALUE
Total Supply	400,000,000 (x10^{18}) Note: the number of decimals is 18, so the total representation token will be 400,000,000 or 400 million.

Table 2. The BigCat token contract properties

2.2.2. Farms contract

Farms contract is a contract that allows investors staking LPToken to get token profit. The owner may create pools with different LPTokens. There is no interaction between these pools.

The contract uses a accRewardPerShare value like an accumulation factor following the time. The updatePoolLastRewardBlock function allows the owner contract updating lastRewardBlock and skipping add the accRewardPerShare value.

2.2.3. Pool contract

Pool contract is also a staking contract like the Farm contract but it only accepts a specific token which owner set.

2.3. Findings

The audit team found some issues in the auditing contracts.

2.3.1. Pools.sol - User can spam updatePool function to add accTokenPerShare unlimited CRITICAL

The contract uses the accRewardPerShare value like an accumulation factor following the time. But the function only updates the lastRewardBlock value every 100 blocks. Therefore, the callers can spam trigger updatePool function to add the accRewardPerShare value unlimited.

```
346  function _updatePool() internal {
347          if (block.number <= lastRewardBlock) {
348              return;
349          }
350
351          uint256 stakedTokenSupply = stakedToken.balanceOf(address(th…
     is));
352
353          if (stakedTokenSupply == 0) {
354              lastRewardBlock = block.number;
355              return;
```

```
356         }
357
358         uint256 multiplier = _getMultiplier(lastRewardBlock, block.n…
    umber);
359         uint256 tokenReward = multiplier.mul(rewardPerBlock);
360         accTokenPerShare = accTokenPerShare.add(
361             tokenReward.mul(PRECISION_FACTOR).div(stakedTokenSupply)
362         );
363         if (block.number >= lastRewardBlock.add(100)) {
364             lastRewardBlock = block.number;
365         }
366         // lastRewardBlock = block.number;
367     }
```

Snippet 1. Pools.sol - User can spam `updatePool` function to add `@accTokenPerShare` unlimited

The lastRewardBlock value is only updated every 100 blocks. In every 100 blocks, the lastRewardBlock value is concreted. Each updatePool function call the accTokenPerShare value has added with the reward which is calculated from lastRewardBlock (constant in 100 blocks) to block.number. It is not true because the number which is used to mul rewardPerBlock is not a constant value. If the caller spam triggers this function, the accumulation factor will be too high and the paid rewardTokens will be a very large number.

RECOMMENDATION

The lastRewardBlock should be updated already the updatePool function is called. With this enhancement, even if how many updatePool function is called, the accTokenPerShare has been added a constant value for an exact amount of time.

```
346 function _updatePool() internal {
347         if (block.number <= lastRewardBlock) {
348             return;
349         }
350
351         uint256 stakedTokenSupply = stakedToken.balanceOf(address(th…
    is));
352
353         if (stakedTokenSupply == 0) {
354             lastRewardBlock = block.number;
355             return;
356         }
357
358         uint256 multiplier = _getMultiplier(lastRewardBlock, block.n…
```

```
umber);
359         uint256 tokenReward = multiplier.mul(rewardPerBlock);
360         accTokenPerShare = accTokenPerShare.add(
361             tokenReward.mul(PRECISION_FACTOR).div(stakedTokenSupply)
362         );
363          lastRewardBlock = block.number;
364     }
```

Snippet 2. Pools.sol - Reccomend fixing

UPDATES

- *Mar 3,2022*: This issue has been acknowledged and fixed by the Darkland team.

2.3.2. Farms.sol - The rewardAmount from withdraw and deposit function may not be equal to the rewardAmount from pendingRewards function LOW

The withdraw function calculates the reward amount like the pendingRewards function but the release token function - safeTokenTransfer, which this function used, does not transfer exactly the amount.

```
294 function withdraw(uint256 _pid, uint256 _amount) public {
295         PoolInfo storage pool = poolInfo[_pid];
296         UserInfo storage user = userInfo[_pid][msg.sender];
297         require(user.amount >= _amount, "withdraw: not good");
298         updatePool(_pid);
299         uint256 pending = user.amount.mul(pool.accRewardPerShare).di…
    v(1e12).sub(
300             user.rewardDebt
301         );
302         safeTokenTransfer(msg.sender, pending);
303         user.amount = user.amount.sub(_amount);
304         user.rewardDebt = user.amount.mul(pool.accRewardPerShare).di…
    v(1e12);
305         pool.lpToken.safeTransfer(address(msg.sender), _amount);
306         emit Withdraw(msg.sender, _pid, _amount);
307     }
```

Snippet 3. Farms.sol - The withdraw function

```
32 function safeTokenTransfer(address _to, uint256 _amount) internal {
33         uint256 rewardBal = rewardToken.balanceOf(address(this));
34         if (_amount > rewardBal) {
35             rewardToken.transfer(_to, rewardBal);
```

```
36          } else {
37              rewardToken.transfer(_to, _amount);
38          }
39      }
```

Snippet 4. Farms.sol - The safeTokenTransfer function release token

If the contract is gonna run out of the reward tokens, the contract only returns the rest balance and updates the state for the users. So the users will be lost tokens and do not have any flow to get the rest tokens.

RECOMMENDATION

We suggest adding a revert statement that sends a message when the contract is run out of the reward tokens. The users can get the reward tokens after the owner added the reward tokens.

UPDATES

- *Mar 3,2022*: This issue has been acknowledged and fixed by the Darkland team.

2.3.3. Pools.sol - Unsafe using transfer and transferFrom method through IERC20 interface LOW

There are some functions in the contract that use transfer, transferFrom methods to call functions from the token contract. With the Bigcat token contract in the audit scope, it doesn't have any problems but the Staking contract allows changing the token contract. So we can't ensure that the transfer and transferFrom function of another token contract works exactly as expected.

For instance, the transfer function can return false with the function call failure instead of returning true or revert like ERC20 Oppenzepplin. With withdraw logic, the user doesn't receive anything while the acc.amount is decreased.

```
168 function withdraw(uint256 _amount) external nonReentrant {
169         UserInfo storage user = userInfo[msg.sender];
170         require(user.amount >= _amount, "Amount to withdraw too high…
    ");
171 
172         _updatePool();
173 
174         uint256 pending = user
175             .amount
176             .mul(accTokenPerShare)
177             .div(PRECISION_FACTOR)
```

```
178                 .sub(user.rewardDebt);
179
180             if (_amount > 0) {
181                 user.amount = user.amount.sub(_amount);
182                 ERC20(stakedToken).transfer(address(msg.sender), _amount…
    );
183             }
184
185             if (pending > 0) {
186                 ERC20(rewardToken).transfer(address(msg.sender), pending…
    );
187             }
188
189             user.rewardDebt = user.amount.mul(accTokenPerShare).div(
190                 PRECISION_FACTOR
191             );
192
193             emit Withdraw(msg.sender, _amount);
194         }
```

Snippet 5. Staking.sol Unsafe using `transfer` method in `withdraw` function

There are four functions that are using them. They are deposit, emergencyWithdraw, recoverWrongTokens and withdraw functions.

RECOMMENDATION

We suggest using SafeERC20 library for IERC20 and changing all transfer, transferFrom method using in the contract to safeTransfer, safeTransferFrom which is declared in SafeERC20 library to ensure that there is no issue when transferring tokens.

UPDATES

- *Mar 3,2022*: This issue has been acknowledged and fixed by the Darkland team.

2.4. Additional notes and recommendations

2.4.1. Farms.sol - Redundant code in safeTokenTransferReward function INFORMATIVE

In the safeTokenTransferReward function, there is a statement that transfers rewardToken from this contract to it. We suggest removing that statement for gas saving.

```
250     function safeTokenTransferReward(uint256 reward) internal {
251             uint256 devReward = reward.div(200).mul(7); // 3.5%
```

```
252         uint256 founderReward = reward.div(200).mul(7); // 3.5%
253         uint256 communityReward = reward.div(25).mul(2); // 8%
254         uint256 totalReward = reward.add(devReward).add(founderRewar…
    d).add(
255             communityReward
256         );
257         uint256 rewardBal = rewardToken.balanceOf(address(this));
258
259         require(rewardBal >= totalReward, "Balance is not enough");
260
261         rewardToken.transfer(devAddr, devReward);
262         rewardToken.transfer(founderAddr, founderReward);
263         rewardToken.transfer(communityAddr, communityReward);
264         rewardToken.transfer(address(this), reward);
265     }
```

Snippet 6. Farms.sol - Redundant code in `safeTokenTransferReward` function

UPDATES

- *Mar 3,2022*: This issue has been acknowledged by the Darkland team.

2.4.2. Unnecessary usage of SafeMath library in Solidity 0.8.0+ INFORMATIVE

All safe math usages in the contract are for overflow checking, solidity 0.8.0+ already do that by default, the only usage of safemath now is to have a custom revert message which isn't the case in the auditing contracts. We suggest using normal operators for readability and gas saving.

Currently, the methods of SafeMath are used in Farms.sol, Pools.sol files.

RECOMMENDATION

We suggest changing all methods from SafeMath library to normal arithmetic operator in the files that we regarded above.

UPDATES

- *Mar 3,2022*: This issue has been acknowledged and fixed by the Darkland team.

APPENDIX

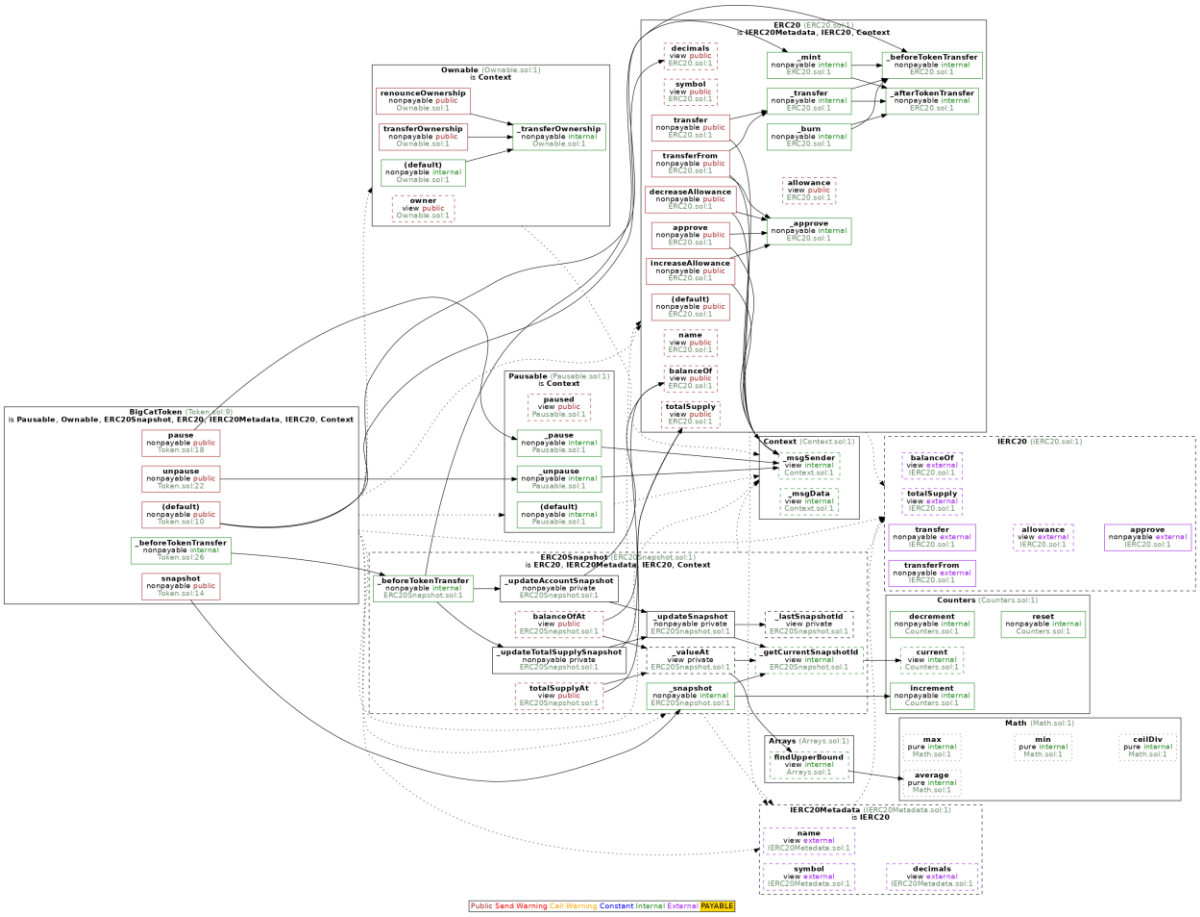

Image 1. BigCat token call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Feb 25, 2022*	Private Report	Verichains Lab
1.1	*Mar 03, 2022*	Public Report	Verichains Lab

Table 3. Report versions history