verichains

SECURITY AUDIT OF

DOT ARCADE SMART CONTRACTS

Public Report

Jan 10, 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.
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 Jan 10, 2022. We would like to thank the Dot Arcade 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 Dot Arcade 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 one vulnerable issue in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Dot Arcade Smart Contracts

DotArcade is the first game mix between Arcade and Moba game genre. Join a multitude of player worldwide and have an opportunity to play to earn NFTs individually or with your clan.Battle: Each battle in DotArcade looks like simple battle in Age of Empires.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Dot Arcade Smart Contracts.

It was conducted on commit ed2a81fe23a4df047148176e8c519f8c7bb01e57 from git repository *https://github.com/DotArcadeNFT/contracts/*.

There are 2 files in our audit scope. They are ADT.sol and ADTVesting.sol files.

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

2.1.1. Contract codes

The Dot Arcade Smart Contracts was written in Solidity language, with the required version to be ^0.8.4.

2.1.1.1. Dot Arcade token contract

This is the contract implement ERC20 token. The source code is referenced from OpenZeppelin's implementation.

Dot Arcade Smart Contracts extends AccessControl, ERC20, ERC20Snapshot and ERC20Pausable contracts. ERC20Snapshot help Token Owner take a snapshot of the balances and total supply at a time for later access. AccessControl allows the contract to implement role-based access control mechanisms which add token owner (contract deployer) OWNER_ROLE role.

Token Owner can pause/unpause contract using ERC20Pausable contract, user can only transfer tokens when contract is not paused.

The contract also supports bot protection from another BP contract which is not in our audit scope.

This table lists some properties of the audited Dot Arcade Smart Contracts (as of the report writing time).

PROPERTY	VALUE
Name	Dot Arcade
Symbol	ADT
Decimals	18
Total Supply	300,000,000 ($x10^{18}$) Note: the number of decimals is 18, so the total representation token will be 300,000,000 or 300 million.

Table 2. The Dot Arcade Token Smart Contract properties

2.1.1.2. Dot Arcade Vesting contract

The Dot Arcade team use this contract to release the tokens that investors bought in the past. The tokens of investors will be released in 2 phases. In the first phase in the TGE time, the investors receive the number of tokens that the Dot Arcade team set. In the second phase, tokens will be released following the linear logic.

2.2. Findings

During the audit process, the audit team found one vulnerability issues in the given version of Dot Arcade Smart Contracts.

2.2.1. ADTVesting.sol - LinearVesting calculate including clifftime HIGH

The vestable function is used to calculate the token releasable with a linear logic. But the function calculates the token releasable from the tgeTime. It seems to be incorrect because it includes the clifftime.

```
function vestable(address beneficiary) public view returns(uint) {
        require(beneficiaries[beneficiary] > 0, "The beneficiary address …
  is invalid");
        require(tge > 0, "TGE is not config");
        uint amount = 0;

        if (block.timestamp > tge) {
            amount = tgeReleases[beneficiary];
        }

        if (block.timestamp < tge + cliffDurations[beneficiary]) {
            return amount - released[beneficiary];
        } else if (block.timestamp >= (tge + durations[beneficiary])) {
            return beneficiaries[beneficiary] - released[beneficiary];
        } else {
            return (beneficiaries[beneficiary] * (block.timestamp - tge) …
  / durations[beneficiary]) - released[beneficiary];
        }
    }
```

Snippet 1. ADTVesting.sol LinearVesting calculate including clifftime

RECOMMENDATION

We suggest changing the function like the below code:

```
function vestable(address beneficiary) public view returns(uint) {
        require(beneficiaries[beneficiary] > 0, "The beneficiary address …
```

```
  is invalid");
        require(tge > 0, "TGE is not config");
        uint amount = 0;

        if (block.timestamp > tge) {
            amount = tgeReleases[beneficiary];
        }

        if (block.timestamp < tge + cliffDurations[beneficiary]) {
            return amount - released[beneficiary];
        } else if (block.timestamp >= (tge + durations[beneficiary])) {
            return beneficiaries[beneficiary] - released[beneficiary];
        } else {
            uint256 vestingAmount = beneficiaries[beneficiary] - tgeRelea…
  ses[beneficiary];
            uint256 currentVestingTime = block.timestamp - tge - cliffDur…
  ations[beneficiary];
            uint256 vestingDuration = durations[beneficiary] - cliffDurat…
  ions[beneficiary];
            return vestingAmount * currentVestingTime/ vestingDuration + …
  tgeReleases[beneficiary] - released[beneficiary];
        }
    }
```

Snippet 2. ADTVesting.sol Reccommend fixing in the `vestable` function

UPDATES

- *Jan 10,2022*: This issue has been acknowledged and fixed by the Dot Arcade team in commit cda3d7fa77c0289a134302afb3bc6b2aba79d46f.

2.3. Additional notes and recommendations

2.3.1. ADTVesting.sol - Unsafe using transfer method through IERC20 interface INFORMATIVE

The release function use transfer method to call function from the token contract. With the ADT token contract in the audit scope, it doesn't have any problems. But if the contract was used for vesing another tokens, it may cause issues in the future development.

For instance with an insecurity contract, the transfer function can return false with the function call failure instead of returning true or revert like ERC20 Oppenzepplin. With release logic, the user doesn't receive anything while the released value still adds.

```
129  function release(address beneficiary) public {
130          require(blockBeneficiaries[beneficiary] == false, "The benef…
     iciary is not exists or blocked");
131          uint vestableAmount = vestable(beneficiary);
132          require(vestableAmount > 0, "The is nothing to vest");
133
134          released[beneficiary] += vestableAmount;
135
136          token.transfer(beneficiary, vestableAmount);
137          emit Released(beneficiary, vestableAmount);
138      }
```

Snippet 3. ADTVesting.sol Unsafe using `transfer` method in `release` function

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

- *Jan 10,2022*: This issue has been acknowledged by the Dot Arcade team.

APPENDIX

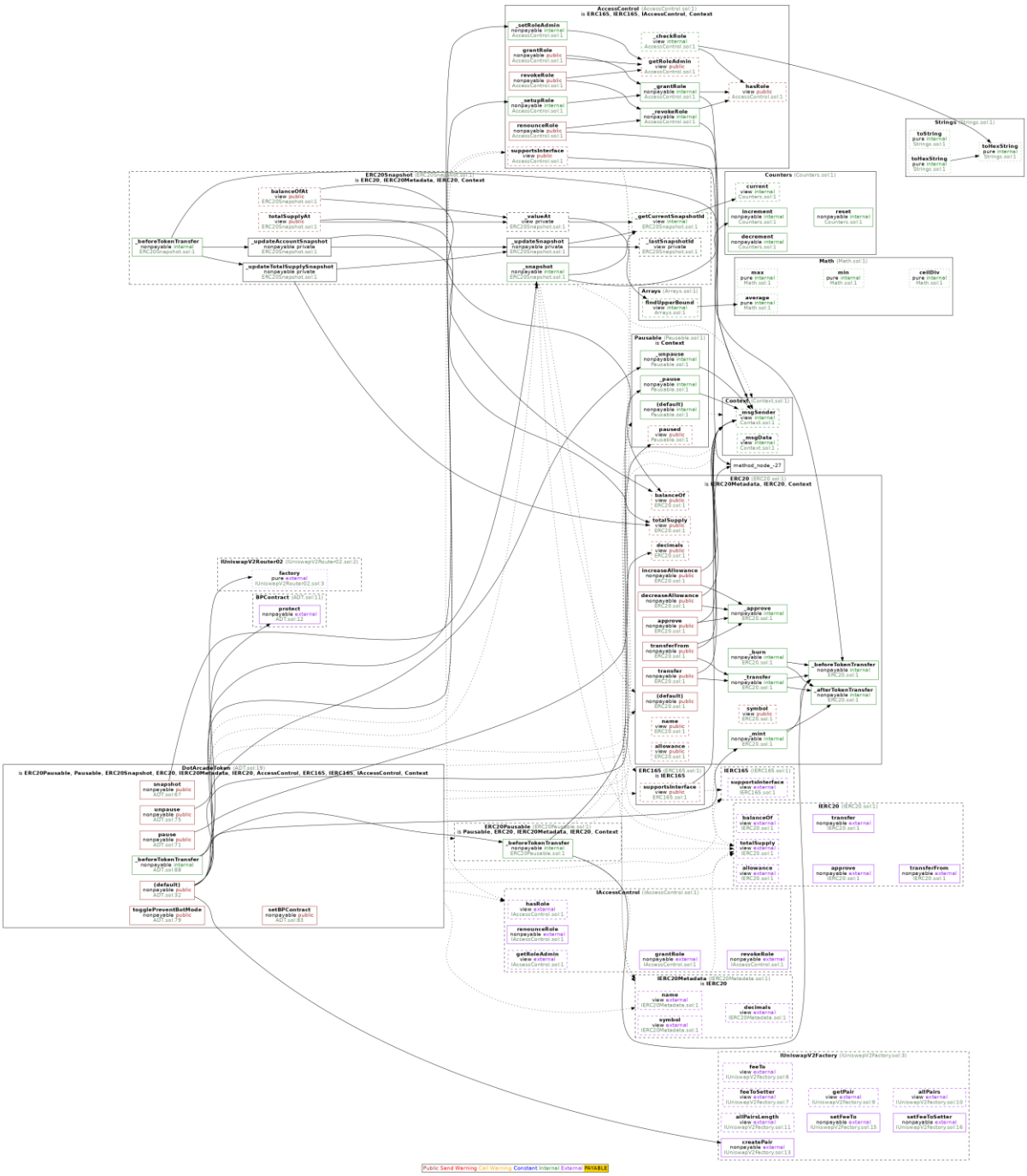

Image 1. Dot Arcade Token Smart Contract call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jan 10, 2022*	Public Report	Verichains Lab

Table 3. Report versions history