DeHacker

Code Security Assessment

# Artyfact - Audit 2

Aug 4th, 2023

# Contents

# Summary

DeHacker's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.
Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow/underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service/logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

# Issue Categories

Every issue in this report was assigned a severity level from the following:

### Critical severity issues

A vulnerability that can disrupt the contract functioning in a number of scenarios or creates a risk that the contract may be broken.

### Major severity issues

A vulnerability that affects the desired outcome when using a contract or provides the opportunity to use a contract in an unintended way.

### Medium severity issues

A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

### Minor severity issues

A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

### Informational

A vulnerability that has informational character but is not affecting any of the code.

# Overview

## Project Summary

| | |
|---|---|
| **Project Name** | Artyfact - Audit 2 |
| **Platform** | BSC |
| **Website** | https://artyfact.game/ |
| **Type** | DeFi,BEP20 |
| **Language** | Solidity |

## Vulnerability Summary

| Vulnerability Level | Total | Mitigated | Declined | Acknowledged | Partially Resolved | Resolved |
|---|---|---|---|---|---|---|
| Critical | 1 | 0 | 0 | 0 | 0 | 1 |
| Major | 1 | 0 | 0 | 0 | 0 | 1 |
| Medium | 0 | 0 | 0 | 0 | 0 | 0 |
| Minor | 1 | 0 | 0 | 0 | 0 | 1 |
| Informational | 1 | 0 | 0 | 0 | 0 | 1 |
| Discussion | 0 | 0 | 0 | 0 | 0 | 0 |

## Audit scope

| ID | File | SHA256 Checksum |
|---|---|---|
| ACK | Artyfact | d743a38254070b5cf6b97085d903341b8a57f59a1b692f40bc0c45f509f651b3 |

# Findings

| ID | Category | Severity | Status |
|---|---|---|---|
| ART-01 | Logical Issue | Critical | Resolved |
| ART-02 | Centralization / Privilege | Major | Resolved |
| ART-03 | Volatile Code | Minor | Resolved |
| ART-05 | Coding Style | Informational | Resolved |

# CRITICAL

## ART-01|PUBLIC burn FUNCTION

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | Critical | ARTY.sol (f3a46e0 ): 287 | Resolved |

## Description

The function burn can be called by anyone to burn any amount of tokens from any other, which shouldn't be allowed.

## Recommendation

Here we provide a test case using Foundry that shows an arbitrary user can call the burn() function to burn tokens of_ultimateOwner .

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "./ARTY.sol";

contract ARTYTest is Test {
    Artyfact public token;
    address account;
    address caller;
    uint256 amount;
    function setUp() public {
        token = new Artyfact();
        caller = address(100); // an arbitrary user
        account = 0x68EF547299A661401aAb3f716D27a29561BE29dB;
        amount = 1000000;
    }

    function testBurn() public {
        vm.startPrank(caller);
        uint256 beforeBalance = token.balanceOf(account);  // balance before burn
        token.burn(account,amount);
        uint256 afterBalance = token.balanceOf(account); // balance after burn
        assertEq(afterBalance + amount, beforeBalance); // True
        vm.stopPrank();
    }
}
```

Consider restricting the function so that only the token owners can burn their own tokens. Or we recommend referencing theOpenZeppelin contract ERC20Burnable to implement the burnFrom function.

```
function burnFrom(address account, uint256 amount) public virtual {
    uint256 currentAllowance = allowance(account, _msgSender());
    require(currentAllowance >= amount, "ERC20: burn amount exceeds allowance");
    unchecked {
        _approve(account, _msgSender(), currentAllowance - amount);
    }
    _burn(account, amount);
}
```

# MINOR

## ART-02|INITIAL TOKEN DISTRIBUTION

| Category | Severity | Location | Status |
|---|---|---|---|
| Centralization / Privilege | Major | ARTY.sol (f3a46e0): 211 | Resolved |

### Description

All ARTY tokens are sent to the contract deployer when deploying the contract. This is a potential centralization risk as thedeployer can distribute ARTY tokens without the consensus of the community.

### Recommendation

We recommend transparency through providing a breakdown of the intended initial token distribution in a public location. Wealso recommend the team make an effort to restrict the access of the corresponding private key.

# MINOR

## ART-03|MISSING ZERO ADDRESS VALIDATION

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | Minor | ARTY.sol (f3a46e0): 138 | Resolved |

### Description

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

```
138          _owner = _tokenOwner;
```

_tokenOwner is not zero-checked before being used.

### Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

# INFORMATIONAL

## ART-05|DECLARATION NAMING CONVENTION

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | Informational | ARTY.sol (f3a46e0 ): 196, 197 | Resolved |

## Description

One or more declarations do not conform to the Solidity style guide with regards to its naming convention.Particularly:

camelCase : Should be applied to function names, argument names, local and state variable names, modifiers
UPPER_CASE : Should be applied to constant variables
CapWords : Should be applied to contract names, struct names, event names and enums

```
196     string private constant _name = 'Artyfact';
```

Constant variable _name is not in UPPER_CASE .

```
197     string private constant _symbol = 'ARTY';
```

Constant variable _symbol is not in UPPER_CASE .

## Recommendation

We recommend adjusting those variable and function names to properly conform to Solidity's naming convention.

# Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

# Appendix

## Finding Categories

### Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

### Coding Style

Coding Style findings usually do not affect the generated bytecode but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

### Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

### Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block. timestamp works.

## Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux “sha256sum" command against the target file.

# About

DeHacker is a team of auditors and white hat hackers who perform security audits and assessments. With decades of experience in security and distributed systems, our experts focus on the ins and outs of system security. Our services follow clear and prudent industry standards. Whether it's reviewing the smallest modifications or a new platform, we'll provide an in-depth security survey at every stage of your company's project. We provide comprehensive vulnerability reports and identify structural inefficiencies in smart contract code, combining high-end security research with a real-world attacker mindset to reduce risk and harden code.

### BLOCKCHAIINS

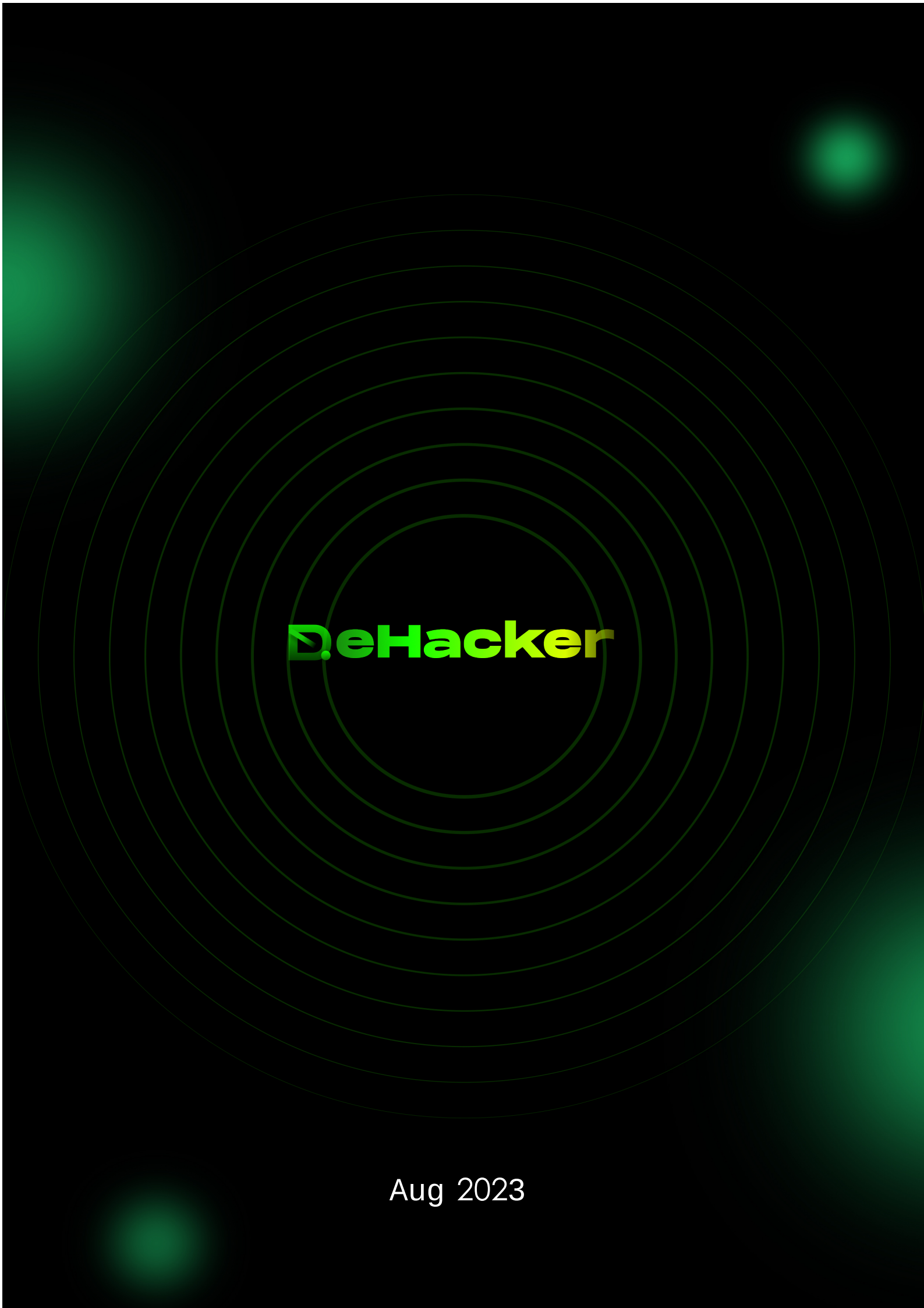
- Ethereum
- Cosmos
- Eos
- Substrate

### TECH STACK

- Python
- Solidity
- Rust
- c++

### CONTACTS

- https://dehacker.io
- https://twitter.com/dehackerio
- https://github.com/dehacker/audits_public
- https://t.me/dehackerio
- https://blog.dehacker.io/




Aug 2023