SecureBlock

Security Audit

Solidity NFT Contract

Prepared for:
Ballistix
www.ballistix.digital

Prepared by:
SecureBlock
www.secureblock.io

1 Introduction

This document includes observations and findings during the security testing of the NFT smart contract requested at the request of the Ballistix client. Security testing was provided by SecureBlock.

a. About SecureBlock

Founded in 2021 by an association of experts in the field of computer security with many years of experience.

Our researchers are continuously working on the development of internal tools and knowledge sharing, as well as by taking and taking recognized certificates in industries such as OSCP, OSCE, OSWE, CEH, CISSP.

Our mission is to simplify and provide a quality security testing service for blockchain projects and technologies. Taking an individual approach and manual review of each project allows us to better understand the use case applications and find vulnerabilities and problems that standard automated tools will not find.

We believe that openness and trust are one of the key aspects of blockchain technology, which is increasingly finding its purpose in more and more industries. For this reason, our clients have an insight into the state of security testing, a preliminary description of vulnerabilities and the public management of the final report through an application we have developed internally.

b. Purpose of the audit

The purpose of the testing was primarily to find security issues, as well as compliance of the code with best practice and, if possible, reduce the gas fee.

2 Executive Summary

a. Results

The conducted testing indicates that the tested application is very safe and follows industry standards as well as best practice ways of development.

A total of 2 security vulnerabilities were found, both of which are classified as informational. Both vulnerabilities relate to the sanity check that is essential to have, but are not mandatory and do not compromise the security of the smart contract itself.

b. Scope

Contract Name	**Aztec Army**
Network	Ethereum - Rinkeby Testnet Network
Language	Solidity
Address	0xfff26060A62204273EA4F877d853a6A646e92155
Filename	AztecArmy.sol
SHA256	51ed8ed355300fed7276f8c9dc49e88a0560b14d9f8c6ecb55a30c0970fd44fc

3 Attack Narative

a. Checklist

In order to find vulnerabilities during the test, we go through a checklist that helps us to cover more tests as well as demonstrate to the client which checks were included during testing. In addition to the list below, we check for business logic vulnerabilities that we find on the deployed contract on our local private network so that there are no unexpected consequences for users.

Name	Description
ERC standards	The contract is using ERC standards.
Compiler Version	The compiler version should be specified.
Constructor Mismatch	The constructor syntax is changed with Solidity versions. Need extra attention to make the constructor function right.
Return standard	Following the ERC20 specification, the transfer and approve functions should return a bool value, and a return value code needs to be added.
Address(0) validation	It is recommended to add the verification of require(_to!=address(0)) to effectively avoid unnecessary loss caused by user misuse or unknown errors
Unused Variable	Unused variables should be removed.
Untrusted Libraries	The contract should avoid using untrusted libraries, or the libraries need to be thoroughly audited too.
Event Standard	Define and use Event appropriately
Safe Transfer	Using transfer to send funds instead of send.

Name	Description
Gas consumption	Optimize the code for better gas consumption.
Deprecated uses	Avoid using deprecated functions.
Sanity Checks	Sanity checks when setting key parameters in the system
Integer overflows	Integer overflow or underflow issues.
Reentrancy	Avoid using calls to trade in smart contracts to avoid reentrancy vulnerability.
Transaction Ordering Dependence	Avoid transaction ordering dependence vulnerability.
Tx.origin usage	Avoid using tx.origin for authentication.
Fake recharge	The judgment of the balance and the transfer amount needs to use the “require function”.
Replay	If the contract involves the demands for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks.
External call checks	For external contracts, pull instead of push is preferred.
Weak random	The method of generating random numbers on smart contracts requires more considerations.
Access Control	Well defined access control for functions.
Authentication management	The authentication management is well defined.

Name	Description
Semantic Consistency	Semantics are consistent
Functionality checks	The functionality is well implemented.

4 Identified Vulnerabilities

Issue ID	Severity	Title	Count
APP-01	Informational	Sanity check in setMaxTokens	1
APP-02	Informational	Sanity check in setSigner	1

APP-01 - Sanity check in setMaxTokens function

It is recommended to add sanity check and make sure that *_maxTokens* parameter in *setMaxTokens* function is greater than 0 and if applicable limit to certain number.

APP-02 - Sanity check in setSigner function

It is recommended to add sanity check and make sure that provided address in the *_signer* variable in *setSigner* function is not equals to zero address.