QuillAudits

Audit Report
January, 2022

For

CRYPTO Baby™
BY OZZY OSBOURNE

Contents

Contents

Scope of the Audit

The scope of this audit was to analyze and document the CryptoBatz Token smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis
Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis
Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis, Theo.

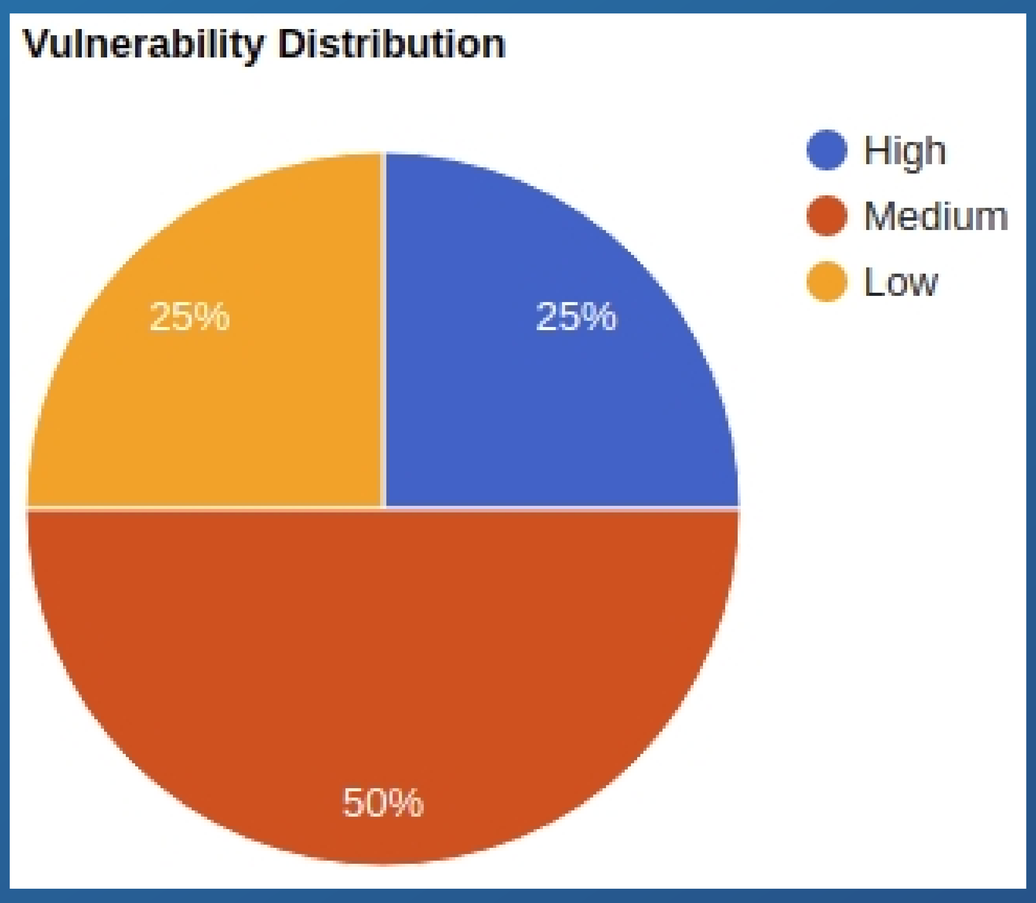
Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	1	1	1	4
Closed	0	1	2	0



Introduction

On **Jan 11, 2021** - QuillAudits Team performed a security audit for **CryptoBatz** smart contracts.

The code for the audit was taken from following the official link: https://github.com/lucid-eleven/crypto-batz-contracts/commit/07ee07dd442c34622798f1ee5b73ef8ff59cef48

V	Date	Commit ID	Files
1	Jan 11	07ee07dd442c34622798f1ee5b73ef8ff59cef48	contracts/*
2	Jan 16	c55d3ee3fc4a471a3c5ab683a7e2111dca617e16	contracts/*

Issues Found – Code Review / Manual Testing

A.Contract - CryptoBatz

High severity issues

1. Random number generation in rollStartIndex function is not random

Line	Code

354-373	function rollStartIndex() external onlyOwner {
	require(PROVENANCE_HASH != 0, "Provenance hash not set");
	require(randomizedStartIndex == 0, "Index already set");
	require(
	block.timestamp >= dutchAuctionConfig.startTime,
	"Too early to roll start index"
	);
	
	uint256 number = uint256(
	keccak256(
	abi.encodePacked(
	blockhash(block.number - 1),
	block.coinbase,
	block.difficulty
	)
	)
	);
	
	randomizedStartIndex = (number % dutchAuctionConfig.supplyLimit) + 1;
	}

Description

The number generated through the function is not random and can be controlled by either the owner or the miner. And can benefit either of the parties in the rarity of NFT.

Remediation

Implement a better random generation scheme, either use a trusted oracle or implement a fair commit-reveal scheme to generate a complete random number.

Status: Acknowledged

Medium severity issues

2. mintAncientBatz function mints NFT with inconsistent tokenId

Line	Code

378-394	function mintAncientBatz(address to, uint256 ancientBatzId) external {
	require(ancientBatzMinter != address(0), "AncientBatz minter not set");
	require(
	msg.sender == ancientBatzMinter,
	"Must be authorized AncientBatz minter"
	);
	require(
	ancientBatzId > 0 && ancientBatzId <= ANCIENT_BATZ_SUPPLY,
	"Invalid AncientBatz Id"
	);
	
	uint256 tokenId = dutchAuctionConfig.supplyLimit + ancientBatzId;
	
	_safeMint(to, tokenId);
	
	totalSupply++;
	}

Description
If ancient bats are minted and **supplyLimit** of **dutchAuctionConfig** is increased, the contract can be bricked, the **buyPublic** will always revert in such cases.

Remediation
Ensure that the **tokenId** minted in the **mintAncientBatz** function isn't inconsistent.

Status: Closed

3. Setter functions of sales in CryptoBatz are not validated properly

Line	Code

420-471	/// @notice Allows the contract owner to update config for the presale
	function configurePresale(
	uint256 startTime,
	uint256 endTime,
	uint256 supplyLimit,
	uint256 mintPrice
	) external onlyOwner {
	uint32 _startTime = startTime.toUint32();
	uint32 _endTime = endTime.toUint32();
	uint32 _supplyLimit = supplyLimit.toUint32();
	
	require(0 < _startTime, "Invalid time");
	require(_startTime < _endTime, "Invalid time");
	
	presaleConfig = PresaleConfig({
	startTime: _startTime,
	endTime: _endTime,
	supplyLimit: _supplyLimit,
	mintPrice: mintPrice
	});
	}
	
	/// @notice Allows the contract owner to update config for the public dutch auction
	function configureDutchAuction(
	uint256 txLimit,
	uint256 supplyLimit,
	uint256 startTime,
	uint256 bottomTime,
	uint256 stepInterval,
	uint256 startPrice,
	uint256 bottomPrice,
	uint256 priceStep
	) external onlyOwner {
	uint32 _txLimit = txLimit.toUint32();
	uint32 _supplyLimit = supplyLimit.toUint32();
	uint32 _startTime = startTime.toUint32();
	uint32 _bottomTime = bottomTime.toUint32();
	uint32 _stepInterval = stepInterval.toUint32();
	
	require(0 < _startTime, "Invalid time");
	require(_startTime < _bottomTime, "Invalid time");

```
dutchAuctionConfig = DutchAuctionConfig({
      txLimit: _txLimit,
      supplyLimit: _supplyLimit,
      startTime: _startTime,
      bottomTime: _bottomTime,
      stepInterval: _stepInterval,
      startPrice: startPrice,
      bottomPrice: bottomPrice,
      priceStep: priceStep
    });
}
```

Description

Owner can configure the sale as he likes, it can be an infinite sale or both sales can be activated at once, thus interfering with each other's supply limit. Moreover, there are no checks on **priceStep**, **startTime**, **bottomTime**, **stepInterval**. Incorrect configuration can brick the ductch auction's **getCurrentAuctionPrice** function.

Remediation

Ensure there are proper validations in these setter functions so the contract might not be bricked or have unintended functionality.

Status: Partially Fixed

Low severity issues

4. Insufficient events

Line	Code
* - *	Across the codebase

Description
Across the codebase, there are important functions such as reserve(), rollStartIndex(), mintAncientBatz(), setBaseURI(), setRoyalties(), setWhitelistSigner()
setAncientBatzMinter(), setProvenance(), configurePresale(), configureDutchAuction() and mint() that don't emit specific events. Whether it’s a crucial setter function or a sale function.

Remediation
Implement and emit specialized events that might help in off-chain monitoring.

Status: Closed

5. function ERC2981.setRoyalties lacks documentation

Line	Code

19-21	function setRoyalties(address recipient, uint256 value) external onlyOwner {
	require(recipient != address(0), "zero address");
	_setRoyalties(recipient, value);
	}

Description

Royalties resetting functionality seems to be undocumented and unclear. Two **SutterTreasury** is being configured, **CryptoBatz** itself acts as a **SutterTreasury** having 3 payees of a split of [50, 45 , 5], another one is deployed when **CryptoBatz** is deployed and have 2 payees of split [70, 30] which being set as primary royalty recipient.

However, the owner of the contract has the power to replace the royalty recipient with any address.

This functionality part of the contract have insufficient documentation regarding how this is intended to work and how it matches with business requirements.

Remediation

Document and test royalty functionality thoroughly.

Status: Closed

6. Inconsistent comment and implementation of reserve() function

Description

The comment @dev above the reserve() function says that "This should be called before presale or public sales start, as only the first MAX_OWNER_RESERVE tokens can be reserved." But in the smart contract there is no require check and the reserve() function can be called after presale or public sales start.

Remediation

It is advised to add the require checks for the same to ensure that this does not happen.

Status: Acknowledged

Informational issues

1. There is missing zero address check for recipient address parameter in the function _setRoyalties(). It is advised to add a require check for the same.

2. Solidity versions: Using very old versions of Solidity prevents benefits of bug fixes and newer security checks. Using the latest versions might make contracts susceptible to
undiscovered compiler bugs. Consider using one of these versions: 0.7.5, 0.7.6 or 0.8.4.

   Refer- https://secureum.substack.com/p/security-pitfalls-and-best-practices-101 and https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

3. The interface defined in IERC2981 is inconsistent with the function defined in ERC2981. The 2nd function parameter for the royaltyInfo function is named as value in ERC2981 whereas in the interface, the same parameter is named as _salePrice. It is advised to resolve these naming conflicts.

4. CryptoBatz contract can be renounced accidentally.Usually, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner, which is risky.

   Remediation- It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

   Refer this post for additional info- https://www.linkedin.com/posts/razzor_github-razzorsecrazzorsec-contracts-activity-6873251560864968705-HOS8

Functional Tests

Test Cases for CryptoBatz.sol

Function Name	Test Result	Logical Result
buyPresale	PASSED	PASSED
buyPublic	PASSED	PASSED
getCurrentAuctionPrice	PASSED	PASSED
tokensOwnedBy	PASSED	PASSED
reserve	ACKNOWLEDGED	PASSED
rollStartIndex	ACKNOWLEDGED	ACKNOWLEDGED
mintAncientBatz	PASSED	PASSED
setBaseURI	PASSED	PASSED
setWhitelistSigner	PASSED	PASSED
setAncientBatzMinter	PASSED	PASSED
setProvenance	PASSED	PASSED
configurePresale	ACKNOWLEDGED	ACKNOWLEDGED
configureDutchAuction	ACKNOWLEDGED	ACKNOWLEDGED
mint	PASSED	PASSED

Test Cases for ERC2981.sol

Function Name	Test Result	Logical Result
_setRoyalties	PASSED	FIXED
royaltyInfo	PASSED	PASSED
supportsInterface	PASSED	PASSED

Test Cases for SutterTreasury.sol

Function Name	Test Result	Logical Result
withdrawAll	PASSED	PASSED

Automated Tests

Slither

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity

Closing Summary

Overall, in the initial audit, there are one critical severity issues associated with random number generation.

No instances of Integer Overflow and Underflow vulnerabilities are found in the contract.

Numerous issues were discovered during the initial audit,some issues are fixed. It is recommended to kindly go through the above-mentioned details and fix the code accordingly.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **CryptoBatz Token**. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **CryptoBatz Token** team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
January, 2022

For

CRYPTO BY OZZY OSBOURNE Ratz™

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com