CryptoBatz
by
Ozzy Osbourne

SMART CONTRACT AUDIT

19.01.2022

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Global Merchandising Services Ltd. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (10.01.2022)	Layout
0.2 (11.01.2022)	Test Deployment
0.5 (11.01.2022)	Automated Security Testing Manual Security Testing
0.6 (12.01.2022)	Testing SWC Checks
0.7 (12.01.2022)	Verify Claims
0.9 (12.01.2022)	Summary and Recommendation
1.0 (12.01.2022)	Final document
1.1 (19.01.2022)	Adding deployed contract address and re-check

2. About the Project and Company

Company address:

Global Merchandising Services Ltd
Matrix Studio Complex
91 Peterborough Road
London SW6 3BU
United Kingdom

Website: https://www.cryptobatz.com

Twitter: https://twitter.com/cryptobatznft

Discord: https://discord.gg/ah54FS98HE

Mail: team@sutter-systems.com

2.1 Project Overview

The collection Crypto Batz introduces a host of unique, innovative features to the NFT world and is the result of a collaboration between Ozzy Osbourne and Sutter Systems. In total there will be 9,666 unique NFT bats, each giving the collector an opportunity to “birth” an additional NFT.

This can be done by activating a feature that will allow the user’s purchase to “bite” and mutate with another NFT from their digital wallet. Known as ‘MutantBatz’, this feature will subsequently allow owners to combine the attributes of two separate projects – making ‘MutantBatz’ a rare offering for NFT collectors. At this time of writing, CryptoBatz will be able to “bite” Bored Ape Yacht Club, SupDucks, CyberKongz, CrypToadz and more to be announced.

In addition to the new NFT range, CryptoBatz is set to launch AncientBatz, a treasure hunt for CryptoBatz holders which will see the virtual Batz scattered around the globe in hidden locations. Each AncientBat will subsequently be able to bite up to 100 times, unlike their CryptoBatz cousin, giving them unrivaled power to breed 100 MutantBatz.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

4.2 Used Code from other Frameworks/Smart Contracts (direct imports)

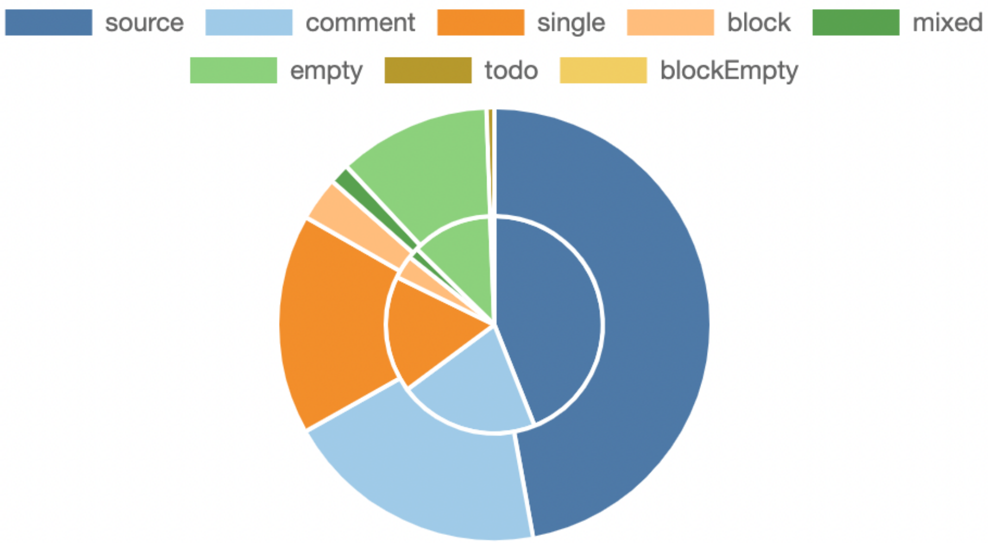
Dependency / Import Path	Source
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/contracts/access/Ownable.sol
@openzeppelin/contracts/finance/PaymentSplitter.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/contracts/finance/PaymentSplitter.sol
@openzeppelin/contracts/token/ERC721/ERC721.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/contracts/token/ERC721/ERC721.sol
@openzeppelin/contracts/utils/Address.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/contracts/utils/Address.sol
@openzeppelin/contracts/utils/cryptography/ECDSA.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/contracts/utils/cryptography/ECDSA.sol
@openzeppelin/contracts/utils/introspection/ERC165.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/contracts/utils/introspection/ERC165.sol
@openzeppelin/contracts/utils/math/SafeCast.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/contracts/utils/math/SafeCast.sol

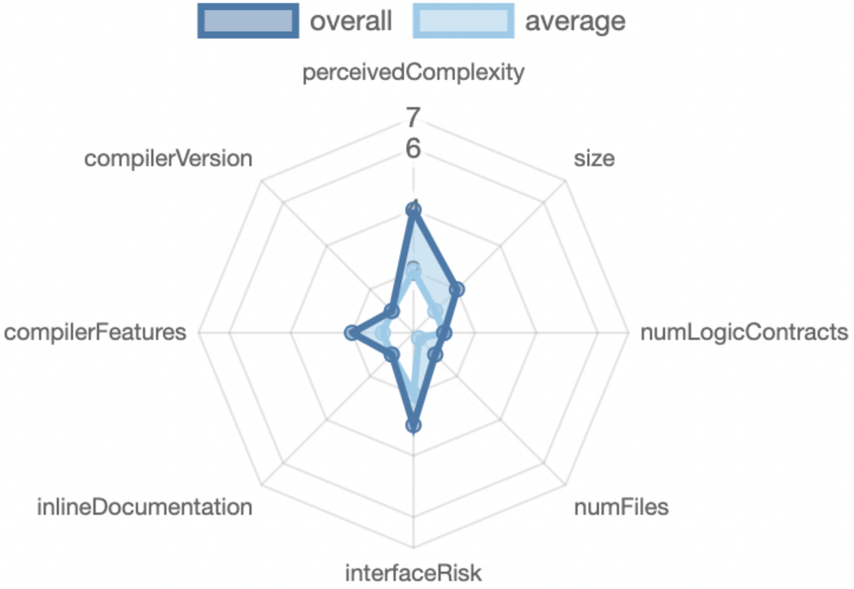
4.3 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	Fingerprint (MD5)
./contracts/CryptoBatz.sol	8798c985985875cb9e255453ce45b69c
./contracts/ERC2981.sol	e031a0ead7dfb457997ac26a1307cab3
./contracts/IERC2981.sol	aa5d4a3425d6e20d0a210375763ec27e
./contracts/SutterTreasury.sol	9120060a69695bd5b2b8998505611fcc

4.4 Metrics / Source Lines & Risk





4.5 Metrics / Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.8.8`		`yes`	`yes` (1 asm blocks)	

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
			`yes`		`yes` → `NewContract:SutterTreasury`

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐Public	💰Payable
19	3

External	Internal	Private	Pure	View
16	20	3	0	7

StateVariables

Total	🌐Public
20	11

4.6 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
📝	contracts/CryptoBatz.sol	1		565	532	310	150	218
📝	contracts/ERC2981.sol	1		48	37	24	7	15
🔍	contracts/IERC2981.sol		1	18	14	3	10	3
📝	contracts/SutterTreasury.sol	1		23	23	17	1	20
📝🔍	**Totals**	**3**	**1**	**654**	**606**	**354**	**168**	**256**

Legend: [➖]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

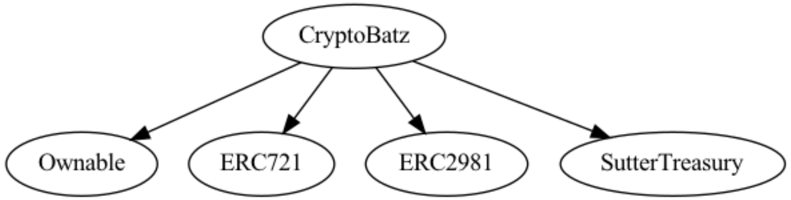
5. Scope of Work

The CryptoBatz Team provided us with the files that needs to be tested. The scope of the audit is the CryptoBatz NFT contract.

The team put forward the following assumptions regarding the security, usage of the contracts:

- The contract is using and ERC standard for NFTs
- Owner cannot mint any new tokens after private sale / ancient batz / owner reserve and public sale minting was done
- Owner cannot burn or lock user NFTs
- Owner cannot pause the contract
- The smart contract is coded according to the newest standards and in a secure way.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.



5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting's experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, Chainsulting's experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES

During the audit, Chainsulting's experts found **no Medium issues** in the code of the smart contract

LOW ISSUES

5.1.1 Missing zero-address check

Severity: LOW
Status: ACKNOWLEDGED
Code: NA
File(s) affected: CryptoBatz.sol

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation, there are several addresses set without checking for the zero address. This can lead to unintended behaviour.	CryptoBatz.setWhitelistSigner(address).newWhitelistSigner (CryptoBatz.sol#405)
	CryptoBatz.setAncientBatzMinter(address).newMinter (CryptoBatz.sol#409)
		We recommend checking addresses for the zero address with require statements before setting them as a variable.

5.1.2 Spelling and Grammatical Errors

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: CryptoBatz.sol

Attack / Description	Code Snippet	Result/Recommendation

Spelling and grammatical errors were identified in the codebase. Fixing these mistakes can help improve the end-user experience by providing clear information on errors encountered and improve the maintainability and auditability of the codebase.	Line: 269
	`"Not enought BATZ remaining"`
		`Better:`
		`"Not enough BATZ remaining"`

INFORMATIONAL ISSUES

5.1.3 A floating pragma is set

Severity: INFORMATIONAL
Status: FIXED
Code: SWC-103
File(s) affected: ALL

Attack / Description	Code Snippet	Result/Recommendation
The current pragma Solidity directive is "^0.7.5". It is recommended to specify a fixed compiler version to	Line 1: `pragma solidity ^0.8.8;`	It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.		i.e. Pragma solidity 0.8.8

5.1.4 State variable could be declared constant

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: CryptoBatz.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation are constant variables not defined as constants.	CryptoBatz.DOMAIN_SEPARATOR (CryptoBatz.sol#130) should be constant CryptoBatz.TYPEHASH (CryptoBatz.sol#129) should be constant	It is recommended to define constant variables properly with the constant keyword to improve code readability.

5.1.5 Storing data via baseURI

Severity: INFORMATIONAL
Status: FIXED
Code: NA
File(s) affected: CryptoBatz.sol

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation the baseURI is not hardcoded or on-chain generated, means the owner/creator is free to choose the way how the metadata file is stored.	Line 396 - 398
	`function setBaseURI(string calldata newBaseUri) external onlyOwner {`
	`    baseURI = newBaseUri;`
	`  }`
		We recommend using IPFS and pinning services to make the metadata behind the baseURI permanently stored.
		To ensure that data persists on IPFS, and is not deleted during garbage collection, data can be pinned to one or more IPFS nodes. Pinning gives you control over disk space and data retention. As such, you should use that control to pin any content you wish to keep on IPFS indefinitely.
		Check more information here: https://docs.ipfs.io/concepts/persistence/#persistence-versus-permanence
		Even if you use an IPFS Service, the file will only exist as long as it is “pinned”. And you still may need a dedicated gateway to serve your files with a decent speed, which may lead to your metadata requests timing out in the future.
		Please look into SVG generated on-chain visuals, for persistent storage.

5.2. SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✅
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✅
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✅
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✅
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✅
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✅
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✅
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✅

ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✅
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✅
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✅
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✅
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✅
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✅

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	☑
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	☑
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	☑
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	☑
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	☑
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	☑
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	☑
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	☑
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	☑
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	☑

ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	X
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	☑
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	☑
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	☑

5.3. Verify Claims

5.3.1 The contract is using and ERC standard for NFTs
Status: tested and verified ✅
The contract is using a correctly implemented ERC-721 and ERC-2981 standard

5.3.2 Owner cannot mint any new tokens after private sale / ancient batz / owner reserve and public sale minting was done
Status: tested and verified ✅

5.3.3 Owner cannot burn or lock user NFTs
Status: tested and verified ✅
There is no burn or lock function

5.3.4 Owner cannot pause the contract
Status: tested and verified ✅
There is no pause function

5.3.5 The smart contract is coded according to the newest standards and in a secure way.
Status: tested and verified ✅

6. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs took place on January 12, 2022.

The main goal of the audit was to verify the claims regarding the security of the smart contract and the functions. During the audit, no critical issues were found after the manual and automated security testing and the claims been successfully verified.

7. Deployed Smart Contract

VERIFIED

https://etherscan.io/address/0xc8adfb4d437357d0a656d4e62fd9a6d22e401aa0#code