Mintoo

NFT Framework

SMART CONTRACT AUDIT

04.04.2023

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 Philippine Digital Asset Exchange (PDAX), Inc. 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 (25.09.2022)	Layout
0.4 (26.09.2022)	Automated Security Testing Manual Security Testing
0.5 (26.09.2022)	Verify Claims and Test Deployment
0.6 (26.09.2022)	Testing SWC Checks
0.9 (27.09.2022)	Summary and Recommendation
1.0 (27.09.2022)	Final document
1.1 (08.10.2022)	Re-check
1.2 (11.11.2022)	Final document
1.3 (16.03.2023)	On-chain verification

2. About the Project and Company

Company address:
Philippine Digital Asset Exchange (PDAX), Inc.
12F Picadilly Star Tower 312 27th St. cor. 4th Ave
BGC, 1634 Taguig City
Philippines

Website: https://mintoo.pdax.ph

Twitter: https://twitter.com/pdaxph

Facebook: https://www.facebook.com/pdaxph

Instagram: https://www.instagram.com/pdaxph

GitHub: https://github.com/PixoPH

LinkedIn: https://www.linkedin.com/company/pdaxph

YouTube: https://www.youtube.com/channel/UCtzLwYb2M_uuWdMYnPAqRfw

Medium: https://medium.com/pdax

Telegram: https://t.me/PDAXCommunity

2.1 Project Overview

PDAX wants every Filipino to have a fair shot at achieving their dreams—no matter where they are in the world and regardless of their social status or financial background.

Having seen the massive digital transformation taking place in the finance industry across the globe, PDAX is aspired to bring this revolution to the Philippine market. Their goal is to make sure every Filipino has the opportunity to invest and participate in the world's future.

Hence, since its founding in 2018, PDAX has led the way in breaking down entry barriers into the field of blockchain and cryptocurrency. They have developed an exchange and launched a mobile app in less than three years to provide Filipinos an edge as they try to build wealth in a highly competitive market. Every day, PDAX strives to bring more value to the Filipino crypto community, whether through educating the public or providing our users with unmatched services.

PDAX is a cryptocurrency exchange, available both as a web-based and mobile app. They give Filipinos the chance to save on fees as they trade the world's leading digital assets in an open market with direct conversion to Philippine Pesos (PHP).

Licensed and regulated by the Bangko Sentral ng Pilipinas, they are fully compliant with the country's laws. And PDAX maintain the highest industry standards for security protocols.

Mintoo is the official digital collectible partner of PDAX

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.

5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 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/ERC1155Collection.sol	9ee7e33ac2d9431041cb95c2fc588fca
./contracts/ERC2981.sol	28ddc5c776a52cabaff9a8cd8e2e4557

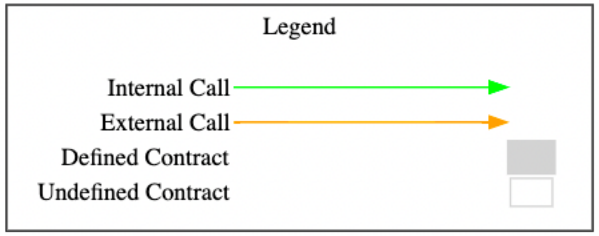
Updated 11/11/2022

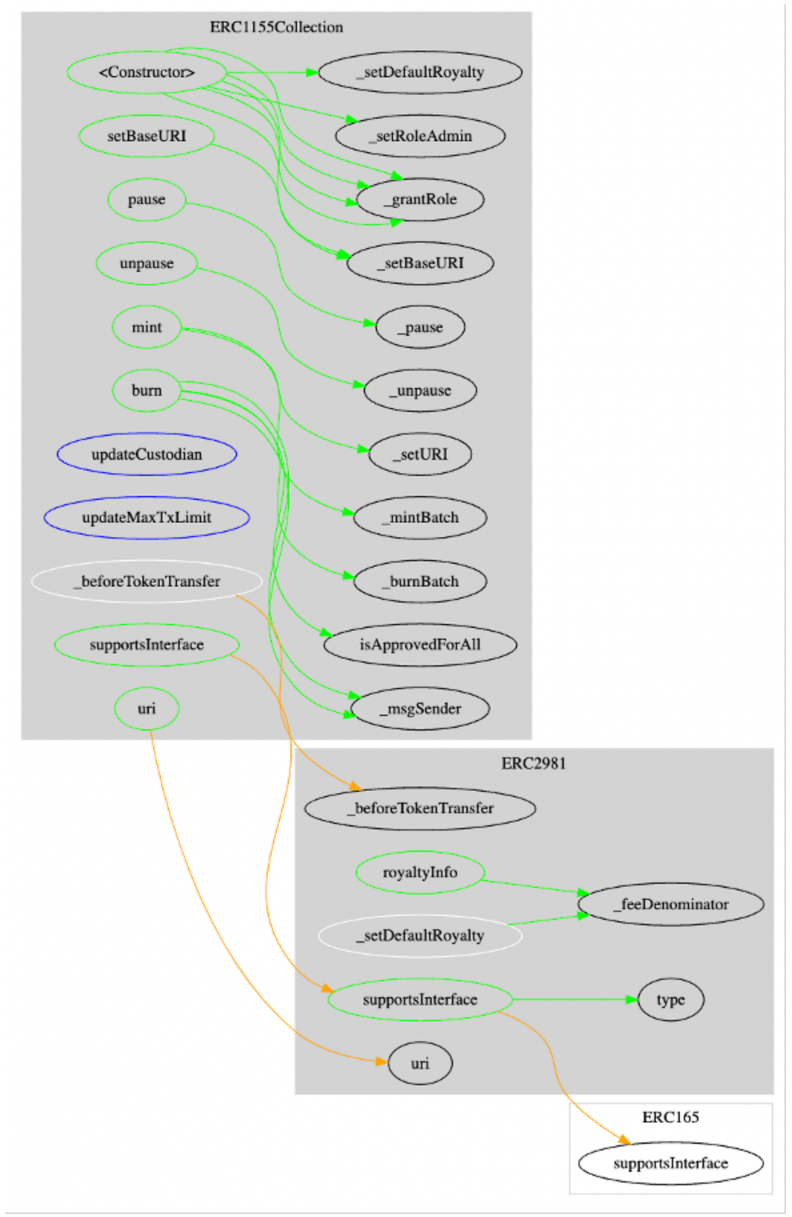
File	Fingerprint (MD5)
./contracts/ERC1155Collection.sol	04ab09ee7a1d380871d9ac9ef1b38206
./contracts/ERC2981.sol	24573a198cad8ca5cdd2914ab9ada1c6

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

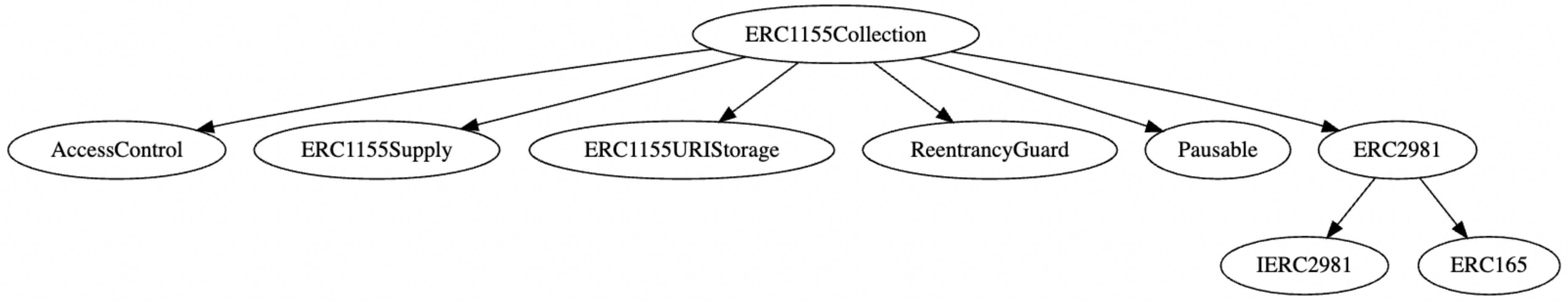
Dependency / Import Path	Source
@openzeppelin/contracts/access/AccessControl.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.7.1/contracts/access/AccessControl.sol
@openzeppelin/contracts/interfaces/IERC2981.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.7.1/contracts/interfaces/IERC2981.sol
@openzeppelin/contracts/security/Pausable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.7.1/contracts/security/Pausable.sol
@openzeppelin/contracts/security/ReentrancyGuard.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.7.1/contracts/security/ReentrancyGuard.sol
@openzeppelin/contracts/token/ERC1155/extensions/ERC1155Supply.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.7.1/contracts/token/ERC1155/extensions/ERC1155Supply.sol
@openzeppelin/contracts/token/ERC1155/extensions/ERC1155URIStorage.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.7.1/contracts/token/ERC1155/extensions/ERC1155URIStorage.sol
@openzeppelin/contracts/utils/introspection/ERC165.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.7.1/contracts/utils/introspection/ERC165.sol

5.3 CallGraph

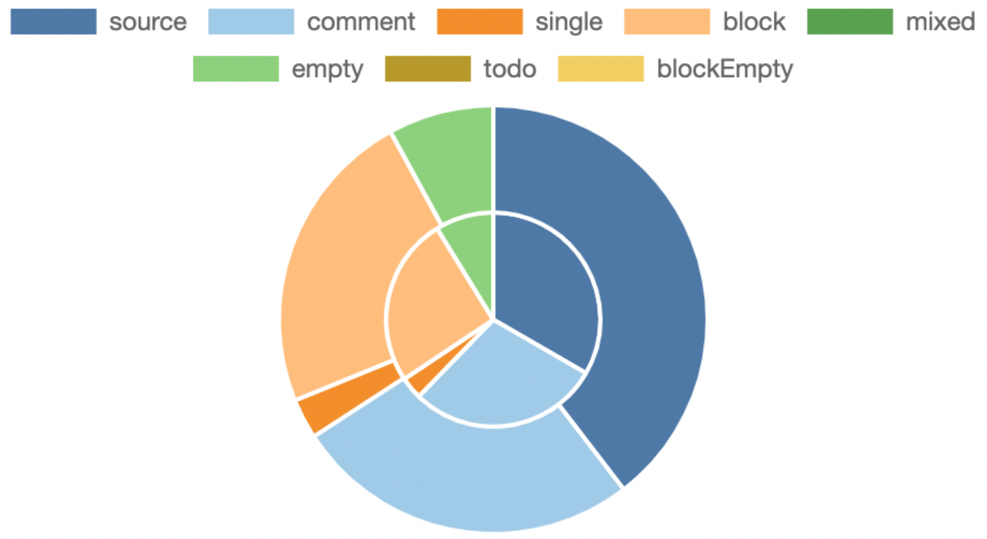


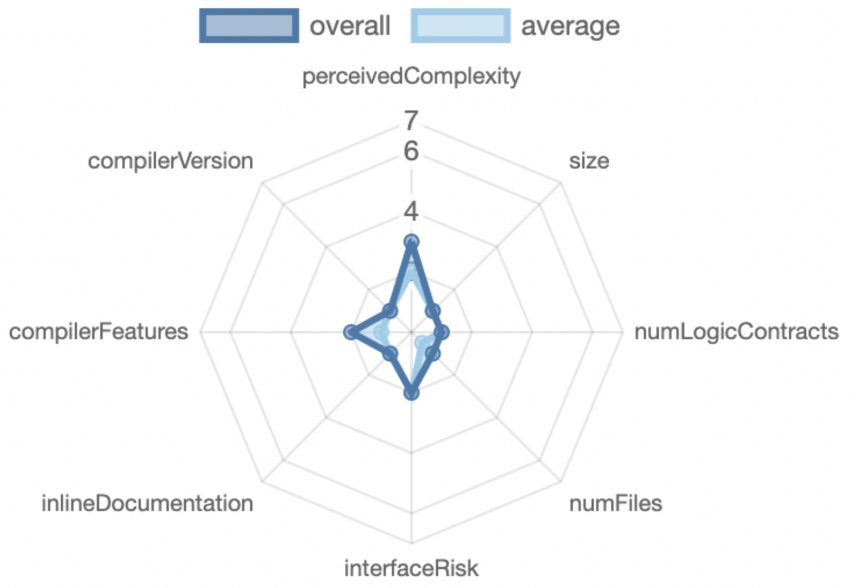


5.4 Inheritance Graph



5.5 Source Lines & Risk





5.6 Capabilities

Solidity Versions observed	🧪 **Experimental Features**	💰 **Can Receive Funds**	🖥 **Uses Assembly**	💣 **Has Destroyable Contracts**
`^0.8.13` `^0.8.0`				

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

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**
11	0

External	**Internal**	**Private**	**Pure**	**View**
2	13	0	1	4

StateVariables

Total	🌐**Public**
10	8

5.7 Source Unites in Scope

Source: https://github.com/PixoPH/nft-smart-contracts
Commit: 4e7c51f148645cb55018e2061301a7eb6d67144e
Branch: main

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	contracts/ERC1155Collection.sol	1		298	255	131	98	105	🧮
🎨	contracts/ERC2981.sol	1		62	62	26	25	18	
📝 🎨	**Totals**	**2**		**360**	**317**	**157**	**123**	**123**	🧮

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, ...)

6. Scope of Work

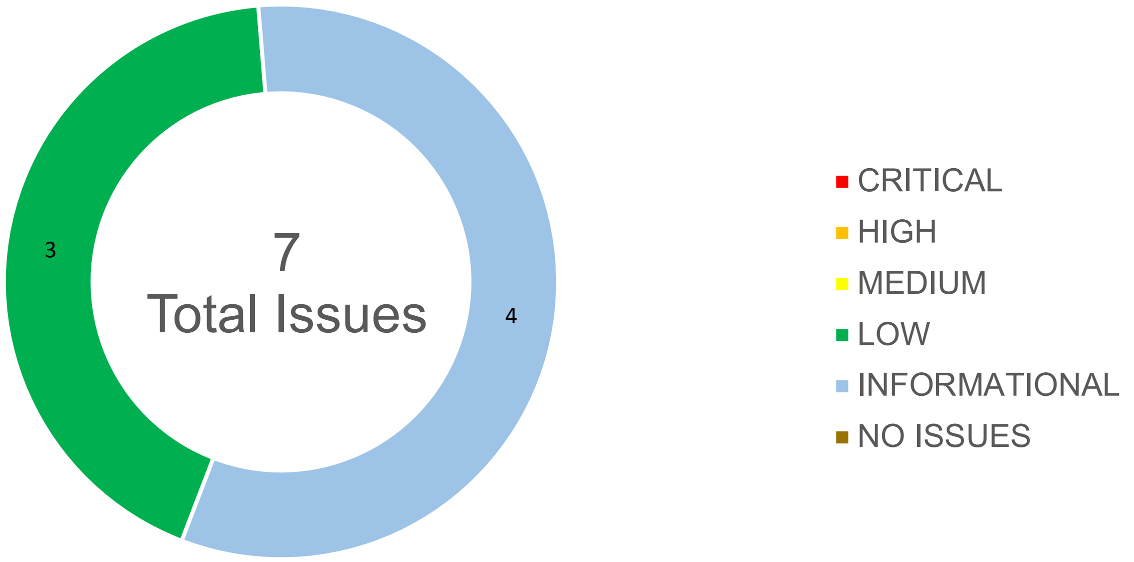
The PDAX Team provided us with the files that needs to be tested. The scope of the audit is the NFT Framework contract.

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

- The ERC-1155 token standard is correctly implemented
- Deployer/Owner cannot mint any new NFTs
- Deployer/Owner cannot burn or lock user funds
- Roles are correctly implemented
- 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.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Missing Value Verification	LOW	FIXED
6.2.2	Misleading In-Line Comment	LOW	FIXED
6.2.3	Gas Optimization	LOW	FIXED
6.2.4	Unused Variable	INFORMATIONAL	FIXED
6.2.5	Naming Convention	INFORMATIONAL	FIXED
6.2.6	Floating Pragma Versions Identified	INFORMATIONAL	FIXED
6.2.7	Storing data via contractURI	INFORMATIONAL	ACKNOWLEDGED

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting‘s experts found **0 Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, Chainsulting’s experts found **0 High issues** in the code of the smart contract.

MEDIUM ISSUES

During the audit, Chainsulting’s experts found **0 Medium issues** in the code of the smart contract.

LOW ISSUES

During the audit, Chainsulting’s experts found **3 Low issues** in the code of the smart contract.

6.2.1 Missing Value Verification

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: ERC1155Collection.sol
Update: commit 2dbc724cc8e095ac973711e4ecf25ad73490f4c2

Attack / Description	The constructor lacks a value safety checks. Therefore, only values that are consistent with the logic of the contract should be permitted. Missing address validation checks could lead to contract deployment without properly set admin, minter or maintainer roles.

Code	Line 55 - 81 (ERC1155Collection.sol)
	`constructor(`
	`        string memory _name,`
	`        string memory _symbol,`
	`        address _creator,`
	`        address _minter,`
	`        address _custodian,`
	`        uint256 _max_tx_mint,`
	`        uint96 _royaltyFee,`
	`        string memory _contractURI,`
	`        string memory _baseURI`
	`    ) ERC1155("") {`
	`        name = _name;`
	`        symbol = _symbol;`
	`        creator = _creator;`
	`        custodian = _custodian;`
	`        maxTxMint = _max_tx_mint;`
	`        contractURI = _contractURI;`
	
	`        _setBaseURI(_baseURI);`
	`        _grantRole(DEFAULT_ADMIN_ROLE, _creator);`
	`        _grantRole(MINTER_ROLE, _minter);`
	`        _grantRole(MAINTAINER_ROLE, _creator);`
	`        _grantRole(MAINTAINER_ROLE, _custodian);`
	`        _setRoleAdmin(MINTER_ROLE, MAINTAINER_ROLE);`
	
	`        _setDefaultRoyalty(creator, _royaltyFee);`
	`    }`

Result/Recommendation	It is recommended to check address values for correctness. This can be done in first stage to exclude zero address in a require statement. In second stage to check if an address is a contract. And most specific for contracts if an address implements a specified interface (EIP-165).

6.2.2 Misleading In-Line Comment

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: ERC1155Collection.sol
Update: commit 9d1a58d787e6e8dbc782bece7cb076ffd760f359

Attack / Description	In the current implementation are some comments not matching to the implemented logic. This could lead to misleading understanding and usage of the code.
Code	Line 147 Array of token ids to be ~~minted~~ burned Line 153 ~~number of tokens to mint must not exceed maxTxMint~~
Result/Recommendation	It is recommended to correct the misleading comments.

6.2.3 Gas Optimization

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: ERC1155Collection.sol
Update: commit 309be785267cffbd963efe48dcd7516a3793be04

Attack / Description	In the current implementation are some dynamic function parameters defined as memory where they could be calldata. Calldata is a non-modifiable and non-persistent data location which consumes less gas than memory while behaving mostly like memory.

Code	Line 89 (ERC1155Collection.sol)
	`function setBaseURI(string memory newBaseUri)`
	`        public`
	`        onlyRole(MAINTAINER_ROLE)`
	`    {`
	`        _setBaseURI(newBaseUri);`
	`    }`
	
	Line 128 – 132 (ERC1155Collection.sol)
	`    function mint(`
	`        uint256[] memory ids,`
	`        uint256[] memory amounts,`
	`        string[] memory cids`
	`    ) public virtual onlyRole(MINTER_ROLE)`
	
	Line 158 – 162 (ERC1155Collection.sol)
	`  function burn(`
	`        address account,`
	`        uint256[] memory ids,`
	`        uint256[] memory values`
	`    ) public virtual`

Result/Recommendation	It is recommended to use calldata instead of memory to reduce gas consumption for contract calls.

INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **4 Informational issues** in the code of the smart contract.

6.2.4 Unused Variable

Severity: INFORMATIONAL
Status: FIXED
Code: NA
File(s) affected: ERC2981.sol
Update: commit b9694f55a6ffb0a95a32a045b5810a02dbc3c01e

Attack / Description	The *_tokenRoyaltyInfo* variable is defined as a private state variable but its value is never initialized. The *_defaultRoyaltyInfo* is used for all tokens.
Code	Line 20 (ERC2981.sol) `mapping(uint256 => RoyaltyInfo) private _tokenRoyaltyInfo;`
Result/Recommendation	It is recommended to remove the unused state variable to reduce gas consumption and enhance code readability. The *royalty* variable in *royaltyInfo* function should be replaced with *_defaultRoyaltyInfo* because there is no other royalty defined at any time.

6.2.5 Naming Convention

Severity: INFORMATIONAL
Status: FIXED
Code: NA
File(s) affected: ERC1155Collection.sol

Update: commit 577e7fd12b8446f010d9fe8e0398ec976f1a2e8d

Attack / Description	In the current implementation immutable variables are written in camel case where they should be written in upper case.
Code	Line 24 (ERC1155Collection.sol) `address public immutable creator;`
Result/Recommendation	It is recommended to write constants and immutable variables in upper case to enhance code readability. Solidity documentation: https://docs.soliditylang.org/en/v0.8.16/style-guide.html#constants

6.2.6 Floating Pragma Versions Identified

Severity: INFORMATIONAL
Status: FIXED
Code: SWC-103
File(s) affected: ALL
Update: commit 22a34e07971f699a4c04664de888a71c5aa998fb

Attack / Description	It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Code	e.g. Line 2
	`pragma solidity ^0.8.13;`
	
	`pragma solidity ^0.8.0;`

Result/Recommendation	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. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version. i.e. pragma solidity 0.8.13 As both contracts using different compiler versions, we would recommend to use only 0.8.13 at both contracts.

6.2.7 Storing data via contractURI

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: ERC1155Collection.sol
Update: As mentioned in the README, "An instance of this smart contract will be [programmatically] deployed ... for each collection in the marketplace." Hence, the contract URI cannot be hardcoded as it differs per collection. We do use a pinning service for saving all contract-level and token-level metadata to IPFS.

Attack / Description	In the current implementation the contractURI is not hardcoded, means the owner/creator is free to choose the way how the metadata file is stored.

Code	Line 30 (ERC1155Collection.sol)
	`// Collection-level metadata for OpenSea`
	`    string public contractURI;`

Result/Recommendation	We recommend using IPFS and pinning services to make the metadata behind the contractURI permanently stored.

6.3 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	✓
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	✓

6.4. Verify Claims

6.4.1 The ERC-1155 token standard is correctly implemented
Status: tested and verified ✅

6.4.2 Deployer/Owner cannot mint any new NFTs
Status: tested and verified ✅

6.4.3 Deployer/Owner cannot burn or lock user funds
Status: tested and verified ✅

6.4.4 Roles are correctly implemented
Status: tested and verified ✅

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

6.5 Unit Tests

```
 Contract: ERC1155Collection
   roles
     ✓ creator has the default admin role (20ms)
     ✓ minter has the minter role (16ms)
     ✓ creator has the maintainer role (31ms)
     ✓ custodian has the maintainer role (21ms)
     ✓ minter role is managed by maintainer role (12ms)
   attributes
     ✓ returns the correct name (17ms)
     ✓ returns the correct symbol (19ms)
     ✓ returns the correct max limit per transaction (15ms)
     ✓ returns the correct contract URI (18ms)
   mint
     ✓ reverts when the number of tokens to mint is greater than the max limit per transaction (1576ms)
     ✓ reverts when the sender is not a minter (94ms)
     mints a single token
       ✓ total supply of minted token increases (19ms)
       ✓ balance of the custodian increases (12ms)
       ✓ emits a transfer batch event (1ms)
     mints multiple tokens
       ✓ total supply of minted token increases (39ms)
       ✓ balance of the custodian increases (52ms)
       ✓ emits a transfer batch event (0ms)
   burn
     ✓ reverts when the caller balance is not enough (25ms)
     ✓ reverts when the caller is not the token owner (22ms)
     burns a single token
```

```
    ✓ total supply of minted token decreases (11ms)
    ✓ balance of the caller decreases (9ms)
    ✓ emits a transfer batch event (1ms)
  burns multiple tokens
    ✓ total supply of minted token decreases (27ms)
    ✓ balance of the caller decreases (34ms)
    ✓ emits a transfer batch event (1ms)
transfer
  ✓ reverts when the sender does not have enough balance (37ms)
  ✓ reverts when the recipient is the zero address (15ms)
  when the sender has enough balance
    ✓ reverts when the caller has not been given approval (17ms)
    when the caller has been given approval
      ✓ transfers the requested amount (52ms)
      ✓ total supply stays the same (26ms)
      ✓ emits a transfer batch event (0ms)
maxTxMint
  ✓ reverts when the sender is not a maintainer (40ms)
  update mint limit per transaction
    ✓ returns the updated limit per transaction (10ms)
    ✓ emits a limit changed event (0ms)
pausing
  when in paused state
    ✓ cannot do token transfers (16ms)
interfaces
  ✓ should support ERC1155 standard (7ms)
  ✓ should support ERC2198 standard (7ms)
royalties
  ✓ should return the correct royalty info when specified (10ms)
  38 passing (12s)
```

6.5.1 Unit Tests Coverage

Test coverage:
- not possible with truffle projects

General Information:
Consider using Hardhat over Truffle for more flexibility during development, more plugins and TypeScript support.

7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase.

The main goal of the audit was to verify the claims regarding the security and functions of the smart contract. During the audit, no critical, no high, no medium, three low and four informational issues have been found, after the manual and automated security testing.

We advise the PDAX team to implement the recommendations to further enhance the code's security and readability.

Update (08/10/2022): the PDAX team addressed all issues and fixed them (https://github.com/PixoPH/nft-smart-contracts/pull/1/commits)

8. Verified Contract

Verified: https://polygonscan.com/address/0xd9c205868754d5b72e49c8f1784c47ad50d266d6#code

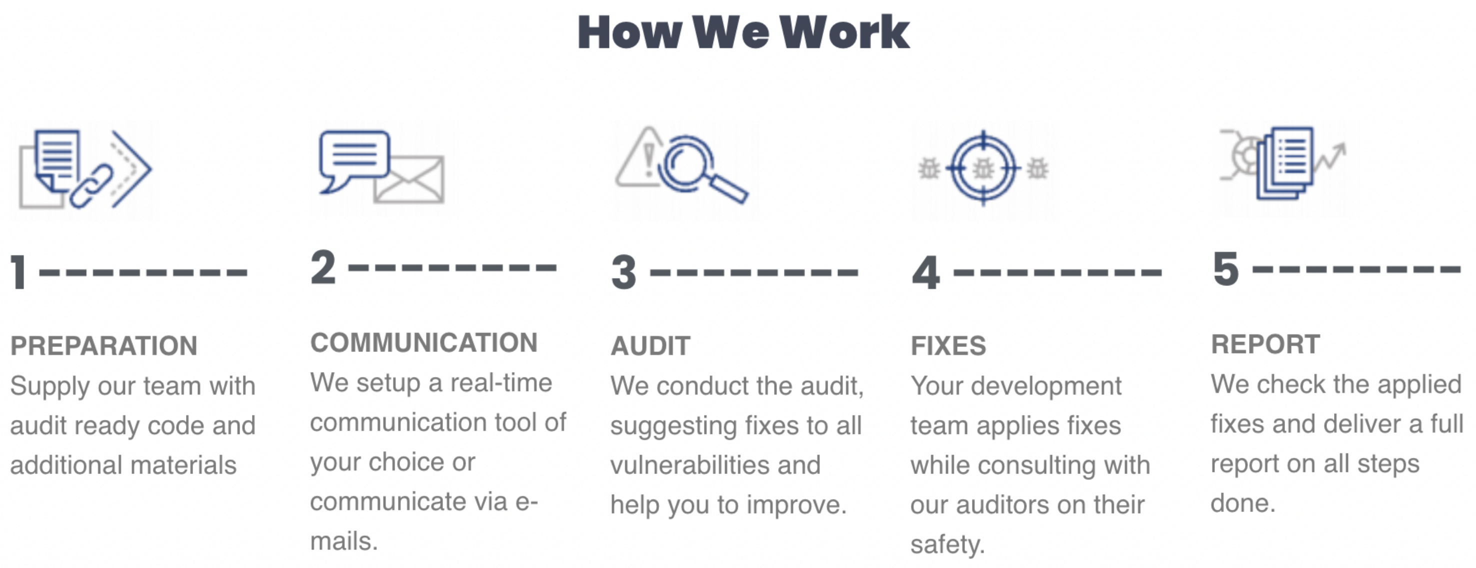
9. About the Auditor

Chainsulting is a professional software development firm, founded in 2017 and based in Germany. They show ways, opportunities, risks and offer comprehensive Web3 solutions. Their services include Web3 development, security and consulting.

Chainsulting conducts code audits on market-leading blockchains such as Solana, Tezos, Ethereum, Binance Smart Chain, and Polygon to mitigate risk and instil trust and transparency into the vibrant crypto community. They have also reviewed and secure the smart contracts of 1Inch, POA Network, Unicrypt, LUKSO among numerous other top DeFi projects.

Chainsulting currently secures $100 billion in user funds locked in multiple DeFi protocols. The team behind the leading audit firm relies on their robust technical know-how in the web3 sector to deliver top-notch smart contract audit solutions, tailored to the clients' evolving business needs.

Check our website for further information: https://chainsulting.de



How We Work

1 --------

PREPARATION
Supply our team with audit ready code and additional materials

2 --------

COMMUNICATION
We setup a real-time communication tool of your choice or communicate via e-mails.

3 --------

AUDIT
We conduct the audit, suggesting fixes to all vulnerabilities and help you to improve.

4 --------

FIXES
Your development team applies fixes while consulting with our auditors on their safety.

5 --------

REPORT
We check the applied fixes and deliver a full report on all steps done.