Tailor NFT

NFT Collectible

SMART CONTRACT AUDIT

21.05.2022

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 TAILOR SAS. 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 (03.05.2022)	Layout
0.4 (04.05.2022)	Automated Security Testing Manual Security Testing
0.5 (05.05.2022)	Verify Claims and Test Deployment
0.6 (07.05.2022)	Testing SWC Checks
0.9 (08.05.2022)	Summary and Recommendation
1.0 (09.05.2022)	Final document
1.1 (15.05.2022)	Re-Check
1.2 (21.05.2022)	Added deployed contract

2. About the Project and Company

Company address:

TAILOR SAS
SIREN 900570144
14 Rue Broca
75005 Paris, France

Website: https://www.tailor-nft.com

Twitter: https://twitter.com/tailor_nft

Discord: https://discord.gg/3gBzEuhHZZ

LinkedIn: https://www.linkedin.com/company/tailor-nft

Instagram: https://www.instagram.com/tailor.nft

Medium: https://medium.com/@tailor-nft

2.1 Project Overview

Tailor builds on NFT tickets to create new experiences around live events. Your tickets will not only allow you to access to an event, it will give you access to a community, allowing early bird tickets for future events, exclusive chats and interview with the artists, exclusive tradable collectibles, immersive online experiences.

Possibilities are endless, thanks to blockchain tech that we make accessible to everyone in a simple and transparent way.

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

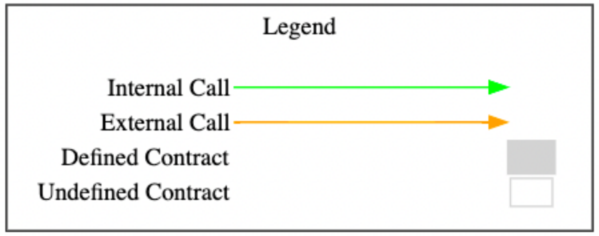
File	Fingerprint (MD5)
./interfaces/ITailorNFTCollectible.sol	a0c43f7913dca9e6086dd2a4018aaea5
./interfaces/ITailorControlAccess.sol	c2e9bbfafe1c888dec34966003b2ad37
./interfaces/IRelayHub.sol	7b6cda20b0f8bd5deec6b29818c12888
./contracts/access/TailorControlAccess.sol	dbef15d7ddb908f9099c955bc430579e
./contracts/token/ERC721/TailorNFTCollectible.sol	132e2057c823bc8dac603e858f4dd47f
./contracts/token/ERC2981/TailorRoyalties.sol	1e669c0c3138bd2d9063f506a9b06557
./contracts/token/libraries/Collections.sol	7c5525527fa166ec7be915f53cfcacf5
./contracts/token/libraries/TokenIdStructure.sol	fcd95ed001f2ab07c8d1046074876e82

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

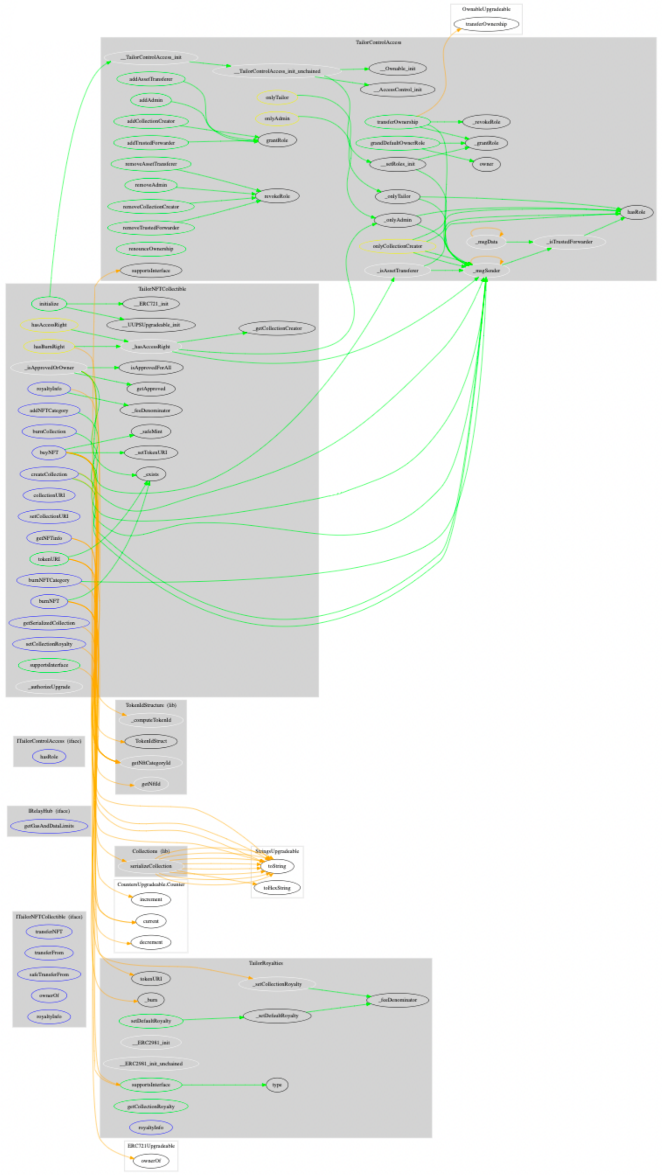
Dependency / Import Path	Source
@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.5.2/access/AccessControlUpgradeable.sol
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.5.2/access/OwnableUpgradeable.sol
@openzeppelin/contracts-upgradeable/interfaces/IERC165Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.5.2/interfaces/IERC165Upgradeable.sol
@openzeppelin/contracts-upgradeable/interfaces/IERC2981Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.5.2/interfaces/IERC2981Upgradeable.sol
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.5.2/proxy/utils/Initializable.sol
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.5.2/proxy/utils/UUPSUpgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721URIStorageUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.5.2/token/ERC721/extensions/ERC721URIStorageUpgradeable.sol
@openzeppelin/contracts-upgradeable/utils/CountersUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.5.2/utils/CountersUpgradeable.sol
@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.5.2/utils/StringsUpgradeable.sol

Dependency / Import Path	Source
@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.5.2/utils/introspection/ERC165Upgradeable.sol
@openzeppelin/contracts/interfaces/IERC165.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.5.0/contracts/interfaces/IERC165.sol

5.3 CallGrap

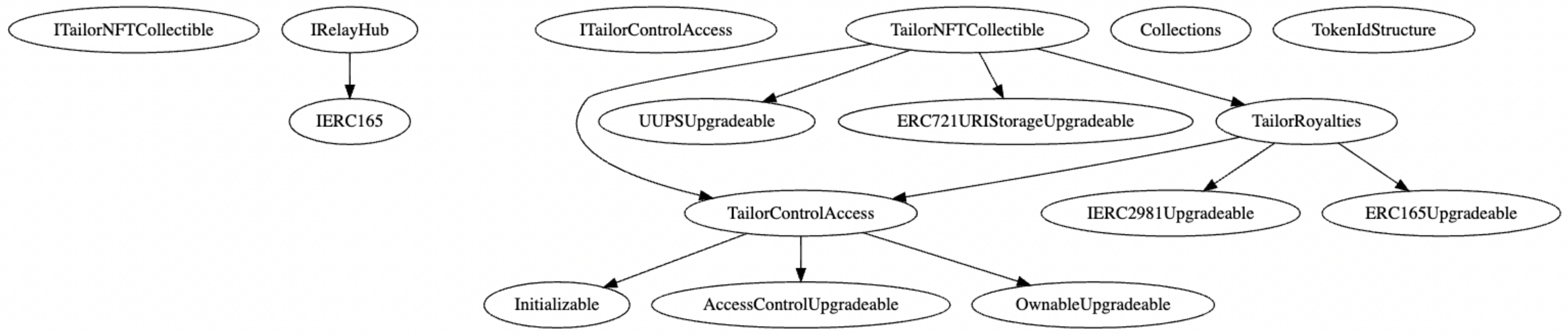


Legend
Internal Call
External Call
Defined Contract
Undefined Contract

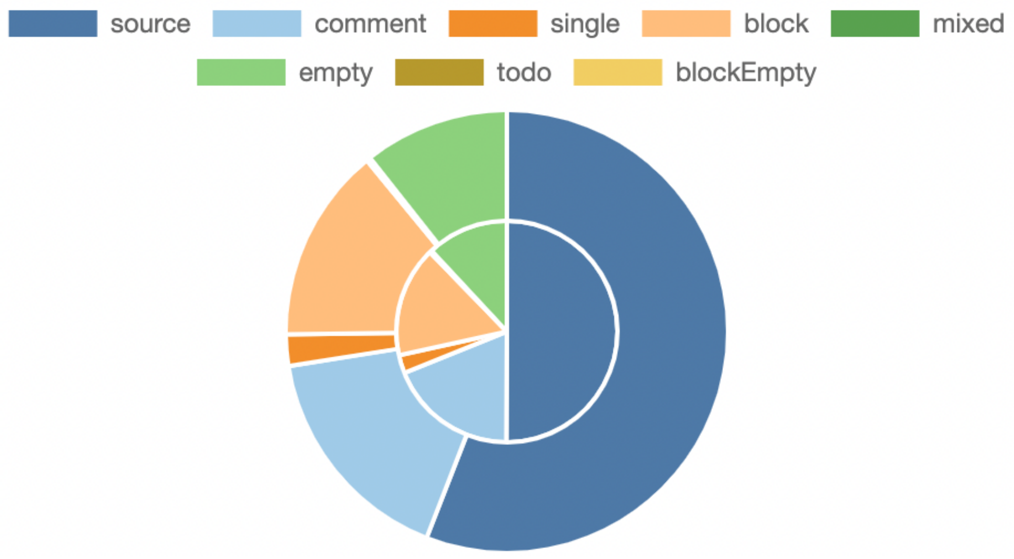


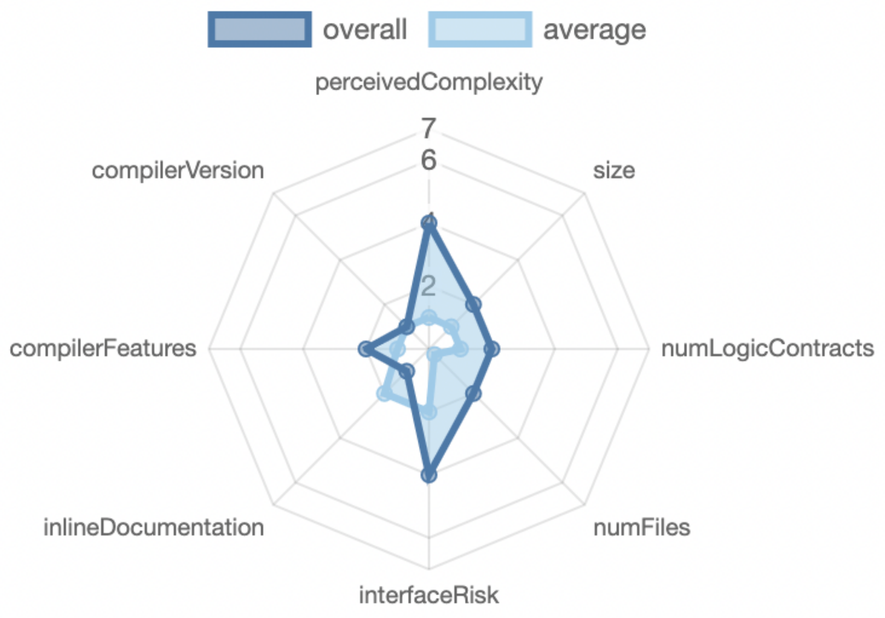
OwnableUpgradeable
TailorControlAccess
TailorNFTCollectible
TokenIdStructure (lib)
ITailorControlAccess (iface)
IRelayHub (iface)
Collections (lib)
StringsUpgradeable
CountersUpgradeable.Counter
ITailorNFTCollectible (iface)
TailorRoyalties
ERC721Upgradeable

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
`^0.8.4` `^0.8.0` `^0.8.9`			`yes` (1 asm blocks)	

📤 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
37	0

External	Internal	Private	Pure	View
20	57	0	4	24

StateVariables

Total	🌐Public
15	0

5.7 Source Unites in Scope

Source: https://github.com/Tailor-NFT/smart-contracts
Last commit: 79d75365ebe561dc3dd1c9902b54619e483280d7

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
🔍	interfaces/ITailorNFTCollectible.sol		1	30	5	3	1	11	
🔍	interfaces/IRelayHub.sol		1	28	24	15	5	5	
🔍	interfaces/ITailorControlAccess.sol		1	6	5	3	1	3	
📝	contracts/access/TailorControlAccess.sol	1		194	174	107	46	99	🖥️🧮
📝	contracts/token/ERC721/TailorNFTCollectible.sol	1		568	477	298	105	165	
📝	contracts/token/ERC2981/TailorRoyalties.sol	1		152	122	70	31	38	
📚	contracts/token/libraries/Collections.sol	1		111	107	92	4	33	
📚	contracts/token/libraries/TokenIdStructure.sol	1		65	57	16	38	5	

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚🔍	Totals	5	3	1154	971	604	231	359	🖥🧮

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

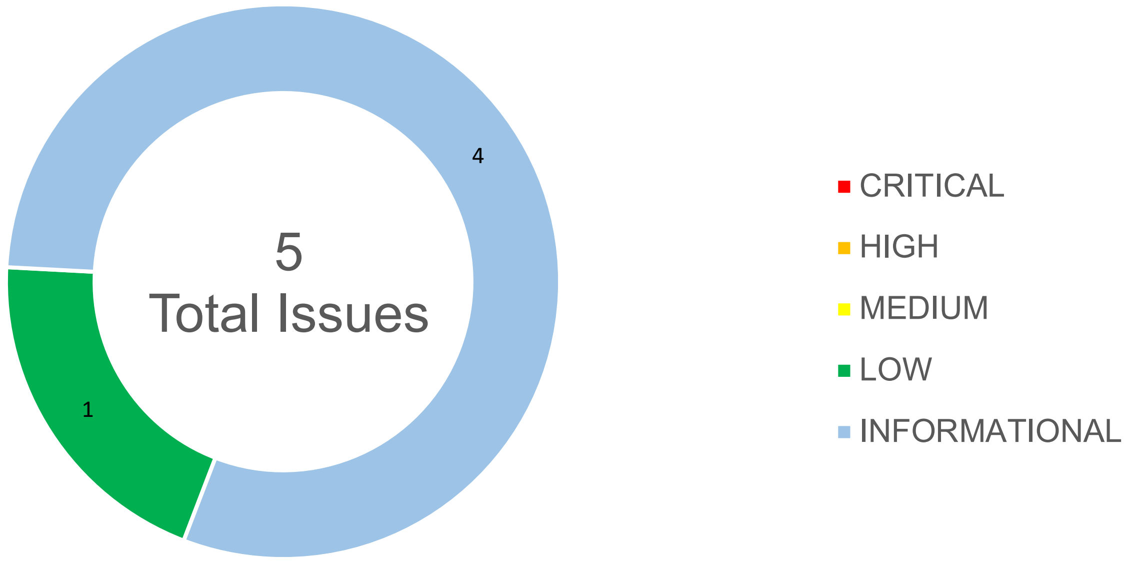
The Tailor NFT Team provided us with the files that needs to be tested. The scope of the audit is the Tailor NFT Collectible contract.

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

- The contract is using and ERC standard for NFTs
- Royalties are correctly implemented and calculated
- 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	Shadowed State Variable	LOW	FIXED
6.2.2	Floating Pragma Version Identified	INFORMATIONAL	FIXED
6.2.3	Dead Code	INFORMATIONAL	FIXED
6.2.4	Failing Unit Tests	INFORMATIONAL	ACKNOWLEDGED
6.2.5	Storing Data Via baseURI	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 **1 Low issue** in the code of the smart contract.

6.2.1 Shadowed State Variable

Severity: LOW
Status: FIXED
Code: SWC-119
File(s) affected: TailorNFTCollectible.sol
Update: 31f523f67d45063cc0d65d872b753adacf2cf153

Attack / Description	In the current implementation, state variables in the deriving contract are shadowed. Unlike functions, state variables cannot be overridden by re-declaring it in the child contract.
Code	Line 76 (TailorNFTCollectible.sol)

	`__ERC721_init(name, symbol);`
	name and symbol are shadowing ERC721Upgradeable state variables.
	
	Line 372, 378, 382, 385 (TailorNFTCollectible.sol)
	
	`string calldata _baseURI,`
	baseURI shadows ERC721Upgradeable state variable.
	
	Line 512 (TailorNFTCollectible.sol)
	
	`address owner = ERC721Upgradeable.ownerOf(tokenId);`
	owner shadows OwnableUpgradeable state variable.

Result/Recommendation	We highly recommend removing the shadowed state variables to avoid unintended behaviour.

INFORMATIONAL ISSUES

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

6.2.2 Floating Pragma Version Identified

Severity: INFORMATIONAL
Status: FIXED
Code: SWC-103
File(s) affected: ALL
Update: 955dbd1a60c2640898b4fe64a34d7f03a5cc2dbe

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	Line 1 `^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 0.8.0

6.2.3 Dead Code

Severity: INFORMATIONAL
Status: FIXED
Code: CWE-561
File(s) affected: TailorNFTCollectible.sol, TailorAccessControl.sol
Update: The code is actually used as it is an override of one of the inherited contracts.

Attack / Description	In the current implementation is some code which is never used. It extends the contract size without any benefits. Unused code is the _isApprovedOrOwner function in TailorNFTCollectible and the ASSET_TRANSFERER role in TailorAccessControl.
Code	Line 502 - 518 (TailorNFTCollectible.sol) `function _isApprovedOrOwner(address spender, uint256 tokenId)` subsequently role ASSET_TRANSFERER is not used Line 20, 77- 79, 96 – 98, 115 – 117 (TailorAccesControl.sol) all functions regarding ASSET_TRANSFERER because it is never used.
Result/Recommendation	We highly recommend removing dead code to reduce contract size and thus gas costs on deployment.

6.2.4 Failing Unit Tests

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: NA
Update: We are aware, we tested a feature which is not currently supported by the deployment lib we are using.

Attack / Description	4 Unit Tests are failing

Code	Log:
	`  151 passing (2m)`
	`  4 failing`
	
	`  1) Upgradeability`
	`       #upgradeProxy - storage collisions`

```
        add state variables to base contract (tailorRoyalties):
     Error: New storage layout is incompatible

contracts/mocks/TestTailorRoyaltiesV2.sol:33: Inserted `royaltytest`
  > New variables should be placed after all existing inherited variables

contracts/mocks/TestTailorRoyaltiesV2.sol:165: Upgraded `__gap` to an incompatible type
  - Bad array resize from 47 to 46
    Size cannot decrease
      at assertStorageUpgradeSafe (node_modules/@openzeppelin/upgrades-core/src/storage/index.ts:35:11)
      at deployImpl (node_modules/@openzeppelin/hardhat-upgrades/src/utils/deploy-impl.ts:108:29)
      at async Proxy.upgradeProxy (node_modules/@openzeppelin/hardhat-upgrades/src/upgrade-proxy.ts:25:32)
      at async upgradeProxy (test/accessControl/Upgradability.test.ts:42:12)
      at async Context.<anonymous> (test/accessControl/Upgradability.test.ts:207:36)

  2) Upgradeability
       #upgradeProxy - storage collisions
         add heritage contract with state variables to NFTCollectible:
     Error: New storage layout is incompatible

contracts/mocks/TestAppendHeritageContract.sol:21: Inserted `testHeritage`
  > New variables should be placed after all existing inherited variables

contracts/mocks/TestAppendHeritageContract.sol:22: Inserted `testMapping`
  > New variables should be placed after all existing inherited variables

contracts/mocks/TestAppendHeritageContract.sol:29: Replaced `__placeholder` with `_structHeritage` of incompatible
type
      at assertStorageUpgradeSafe (node_modules/@openzeppelin/upgrades-core/src/storage/index.ts:35:11)
      at deployImpl (node_modules/@openzeppelin/hardhat-upgrades/src/utils/deploy-impl.ts:108:29)
      at async Proxy.upgradeProxy (node_modules/@openzeppelin/hardhat-upgrades/src/upgrade-proxy.ts:25:32)
      at async upgradeProxy (test/accessControl/Upgradability.test.ts:42:12)
      at async Context.<anonymous> (test/accessControl/Upgradability.test.ts:250:36)

  3) Upgradeability
       verify state of proxy after upgrade
         should upgrade (appendToBase) and still have the state in the proxy :
     Error: New storage layout is incompatible
```

```
contracts/mocks/TestTailorRoyaltiesV2.sol:33: Inserted `royaltytest`
  > New variables should be placed after all existing inherited variables

contracts/mocks/TestTailorRoyaltiesV2.sol:165: Upgraded `__gap` to an incompatible type
  - Bad array resize from 47 to 46
    Size cannot decrease
      at assertStorageUpgradeSafe (node_modules/@openzeppelin/upgrades-core/src/storage/index.ts:35:11)
      at deployImpl (node_modules/@openzeppelin/hardhat-upgrades/src/utils/deploy-impl.ts:108:29)
      at async Proxy.upgradeProxy (node_modules/@openzeppelin/hardhat-upgrades/src/upgrade-proxy.ts:25:32)
      at async upgradeProxy (test/accessControl/Upgradability.test.ts:42:12)
      at async Context.<anonymous> (test/accessControl/Upgradability.test.ts:306:36)

  4) Upgradeability
       verify state of proxy after upgrade
         should upgrade (addHeritage) and still have the state in the proxy :
     Error: New storage layout is incompatible

contracts/mocks/TestAppendHeritageContract.sol:21: Inserted `testHeritage`
  > New variables should be placed after all existing inherited variables

contracts/mocks/TestAppendHeritageContract.sol:22: Inserted `testMapping`
  > New variables should be placed after all existing inherited variables

contracts/mocks/TestAppendHeritageContract.sol:29: Replaced `__placeholder` with `_structHeritage` of incompatible
type
      at assertStorageUpgradeSafe (node_modules/@openzeppelin/upgrades-core/src/storage/index.ts:35:11)
      at deployImpl (node_modules/@openzeppelin/hardhat-upgrades/src/utils/deploy-impl.ts:108:29)
      at async Proxy.upgradeProxy (node_modules/@openzeppelin/hardhat-upgrades/src/upgrade-proxy.ts:25:32)
      at async upgradeProxy (test/accessControl/Upgradability.test.ts:42:12)
      at async Context.<anonymous> (test/accessControl/Upgradability.test.ts:312:36)



error Command failed with exit code 4.
```

Result/Recommendation	Check these test cases and fix them to ensure full test coverage

6.2.5 Storing Data Via baseURI

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: TailorNFTCollectible.sol
Update: We intend to use IPFS (or another storage service like that) later on, but not right now as we need to learn more about what / how we want to store metadata/NFT data

Attack / Description	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.
Code	Line 370 - 386 (TailorNFTCollectible.sol)

```
  function setCollectionURI(
        uint128 collectionId,
        string calldata _baseURI,
        string calldata _urimetadata
    ) external hasAccessRight(collectionId) {
        Collections.Collection storage c = collections[collectionId];
        require(bytes(c.baseURI).length > 0, "Collection not found");
        require(
            (bytes(_baseURI).length > 0) && (bytes(_urimetadata).length > 0),
            "InvalidURIError"
        );

        c.baseURI = _baseURI;
        c.metadataURI = _urimetadata;

        emit CollectionURIChanged(collectionId, _baseURI, _urimetadata);
    }
```

Result/Recommendation	We recommend using IPFS and pinning services to make the metadata behind the baseURI permanently stored or implement into the event creation service.

	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 Keep in mind even if you use an IPFS Service, the file will only exist as long if 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 investigate SVG generated on-chain visuals and on-chain stored metadata, for persistent storage.

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	✓

ID	Title	Relationships	Test Result
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	✓
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	X

ID	Title	Relationships	Test Result
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')	✓
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	✓

ID	Title	Relationships	Test Result
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	✅
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	✅

6.4. Verify Claims

6.4.1 The contract is using and ERC standard for NFTs
Status: tested and verified ✅

6.4.2 Royalties are correctly implemented and calculated
Status: tested and verified ✅

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

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, 1 low and 4 informational issues have been found, after the manual and automated security testing. No necessary need for action, as the recommendations only further enhance the code's readability, not security.

8. Deployed Smart Contract

VERIFIED

Proxy: https://polygonscan.com/address/0x7D202bD4759521CEfccCd27120b073AaaeB16733#code

Implementation: https://polygonscan.com/address/0xb7eec13aff29c592a59af3f6b96fa6c6e9dd2b1a#code

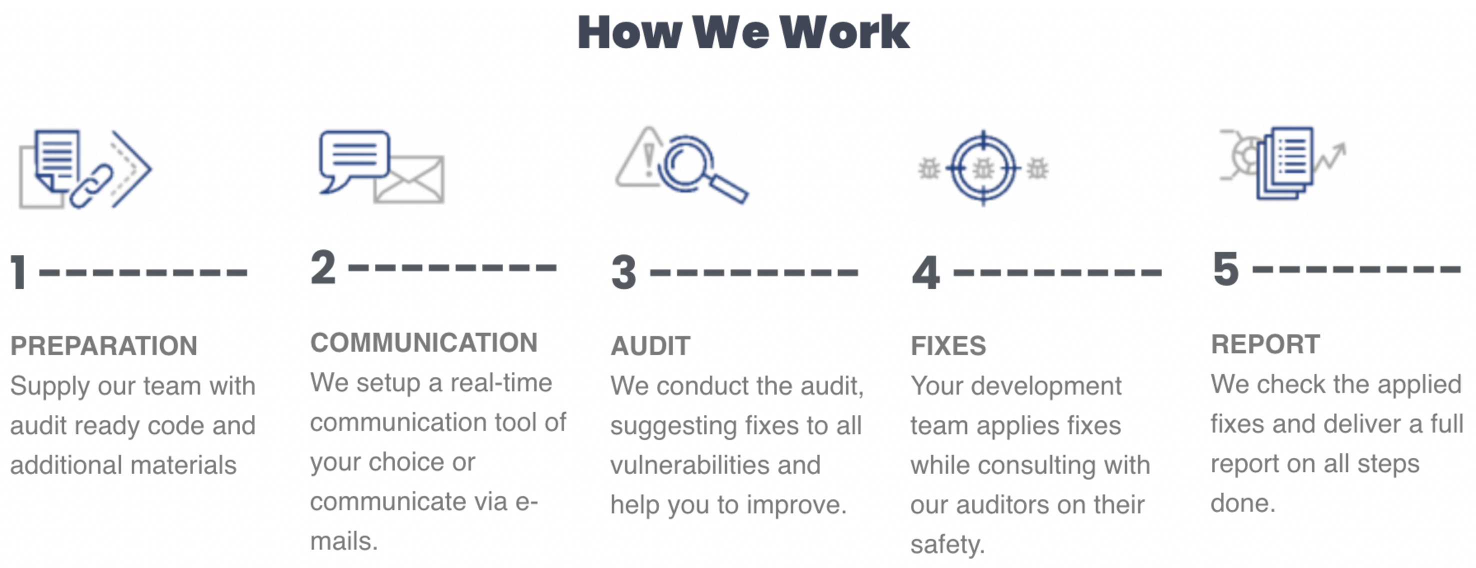
9. About the Auditor

Chainsulting is a professional software development firm, founded in 2017 and based in Germany. They show ways, opportunities, risks and offer comprehensive blockchain solutions. Some of their services include blockchain development, smart contract audits and consulting.

Chainsulting conducts code audits on market-leading blockchains such as Hyperledger, Tezos, Ethereum, Binance Smart Chain, and Solana 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, Amun, Furucombo 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 blockchain 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.