



Capsule Corp. Labs

20 Mint NFT

SMART CONTRACT AUDIT

26.04.2022

Made in Germany by Chainsulting.de



Table of contents

1. Disclaimer.....	4
2. About the Project and Company	5
2.1 Project Overview.....	6
3. Vulnerability & Risk Level	7
4. Auditing Strategy and Techniques Applied.....	8
4.1 Methodology	8
5. Metrics	9
5.1 Tested Contract Files	9
5.2 Used Code from other Frameworks/Smart Contracts	10
5.3 CallGraph.....	11
5.4 Inheritance Graph	12
5.5 Source Lines & Risk.....	13
5.6 Capabilities	14
5.7 Source Unites in Scope	15
6. Scope of Work.....	16
6.1 Findings Overview	17
6.2 Manual and Automated Vulnerability Test.....	18
6.2.1 Spelling And Grammatical Errors	18
6.2.2 Missing Zero Address Checks	19
6.2.3 Renounce Ownership	20
6.2.4 Floating Pragma Version Identified	21
6.2.5 Storing Data Via baseURI.....	21



6.3 SWC Attacks	23
6.4. Verify Claims	27
7. Executive Summary.....	28
8. Deployed Smart Contract	28
9. About the Auditor	29



1. Disclaimer

The audit makes no statements or warranties 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 CAPSULE CORP. LABS SASU. 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 (23.04.2022)	Layout
0.4 (23.04.2022)	Automated Security Testing Manual Security Testing
0.5 (24.04.2022)	Verify Claims and Test Deployment
0.6 (24.04.2022)	Testing SWC Checks
0.9 (25.04.2022)	Summary and Recommendation
1.0 (25.04.2022)	Final document
1.1 (26.04.2022)	Deployed contracts

2. About the Project and Company

Company address:

CAPSULE CORP. LABS SASU
SIREN 902140037
19 Bd Malesherbes
75008 Paris, France



Capsule Corp.

Website: <https://www.capsule-corp.io>

Twitter: <https://twitter.com/CapsuleCorpLabs>

Medium: <https://medium.com/capsule-corp-labs>

LinkedIn: <https://www.linkedin.com/company/capsule-corp-labs>

20Mint

Website: <https://20mint.xyz>

Twitter: <https://twitter.com/20MintFR>

Discord: <https://discord.gg/kzjrDvK6>



2.1 Project Overview

Capsule Corp. Labs partners with developers of Augmented NFT solutions by providing them with all the resources they need for the development of their decentralized application, from ideation to launch on the Ternoia Chain.

An Augmented NFT is a Non-Fungible Token enriched with virtual components that communicate with the user's environment at the user's request. Capsule Corp Labs provides with all the resources needed to build, launch, run and monetize a decentralized application for Augmented NFTs, whether it pertains to collectibles, entertainment, gaming or the metaverse.

The 20 Min NFT contract is created for 20 minutes (pronounced vingt minutes) a free, daily newspaper aimed at commuters in France. It is published by Schibsted and Ouest-France Group. 20 minutos, the Spanish version, is distributed by Schibsted and Zeta in Spain. In Switzerland, the French-language edition 20 minutes and the German-language edition 20 Minuten are published by Tamedia. In 2017, it claimed that its website received 16 million unique users per month. In Greater Paris, Ipsos and CESP confirmed a circulation of 805,000 with a readership of 2,339,000. 20 minutes claims that its readers are "young urban citizens (15–40 years old) that to a lesser extent consume traditional newspapers."

The French 20 minutes was launched in Paris on 15 March 2002, and spread to 11 other urban areas of France, including, in order of size, the cities of Marseille, Lyon, Toulouse, Nice, Nantes, Strasbourg, Montpellier, Bordeaux, Lille, Rennes and Grenoble. Each edition includes both national pages and regional sections. Since its launch, 20 minutes has led the market of free French newspapers. In March 2014, due to the fall of advertising revenues (-6% en 2013), TF1 and Bolloré, owners of 20 minutes' competitors—Metronews and Direct Matin—, announced their willingness to buy 20 minutes and merge their activities. The name 20 minutes refers to the amount of time it should take one to read this daily newspaper.

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

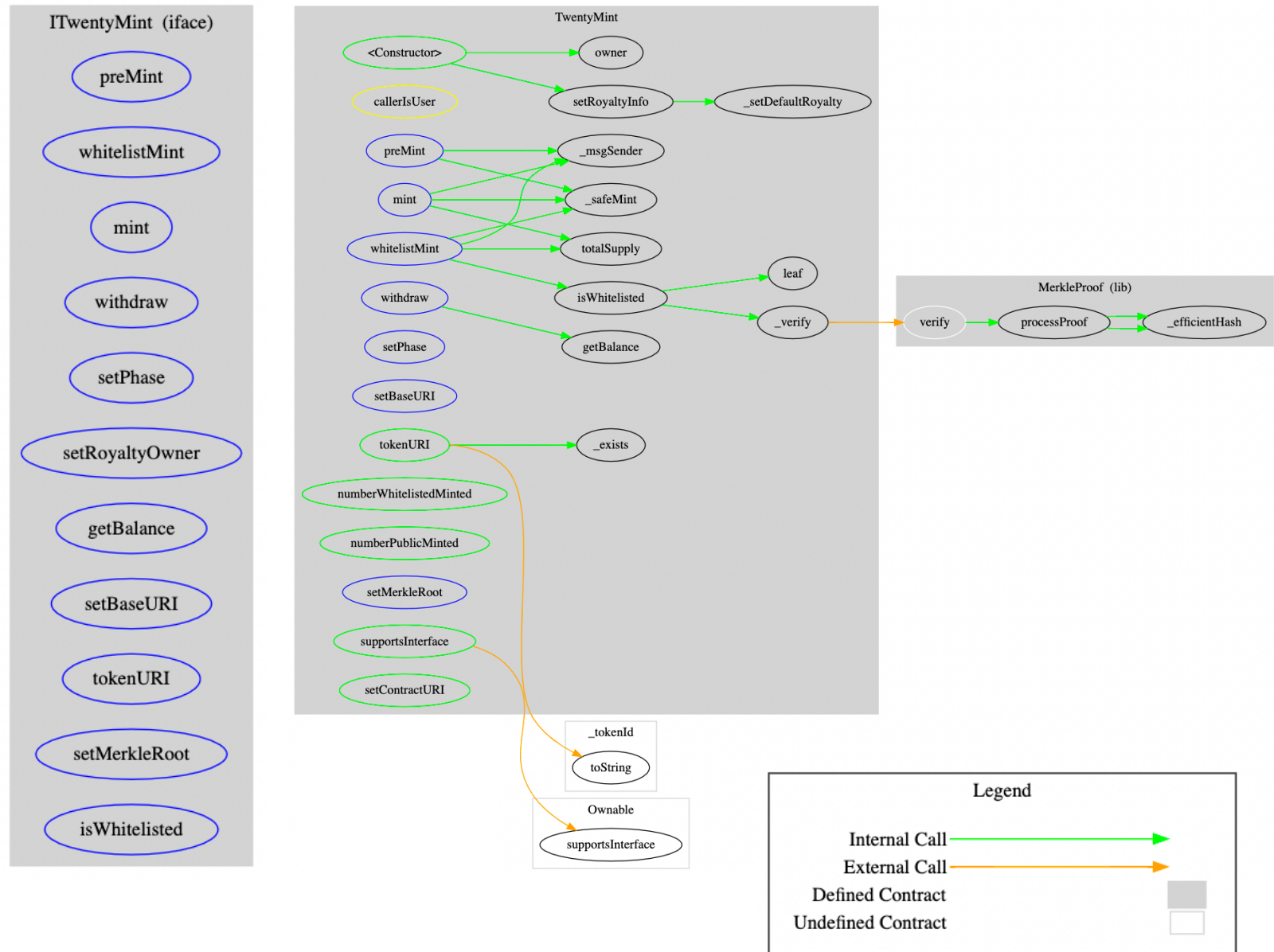
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability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

File	Fingerprint (MD5)
./contracts/interfaces/ITwentyMint.sol	bb27578c7f14455e7756a10a3c4a3b9f
./contracts/TwentyMint.sol	d12f589ebecb3c94061723d37ac152dc
./contracts/utis/MerkleProof.sol	b2d1ac0bec7a8df12c3bf0d51d4b2dca

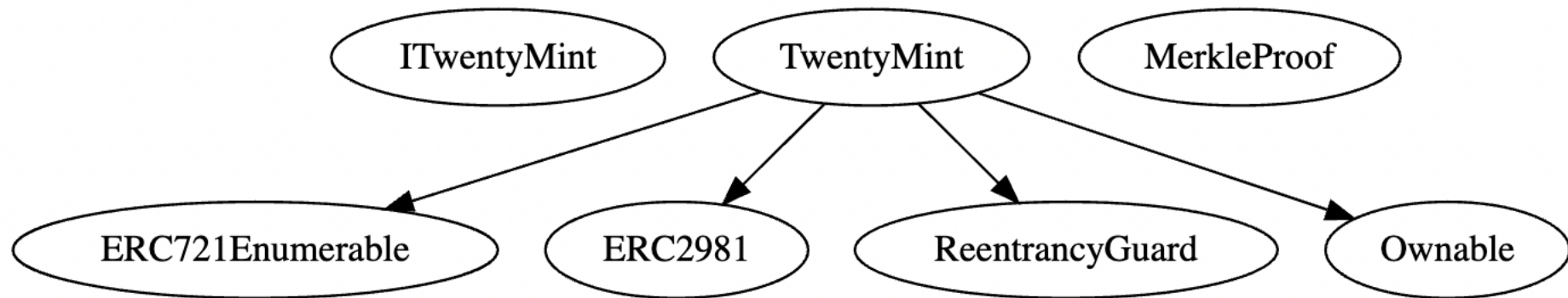
5.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.5.0/contracts/access/Ownable.sol
@openzeppelin/contracts/security/ReentrancyGuard.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.5.0/contracts/security/ReentrancyGuard.sol
@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.5.0/contracts/token/ERC721/extensions/ERC721Enumerable.sol
@openzeppelin/contracts/token/ERC721/extensions/ERC721Royalty.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.5.0/contracts/token/ERC721/extensions/ERC721Royalty.sol
@openzeppelin/contracts/token/common/ERC2981.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.5.0/contracts/token/common/ERC2981.sol
@openzeppelin/contracts/utils/Strings.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.5.0/contracts/utils/Strings.sol

5.3 CallGraph

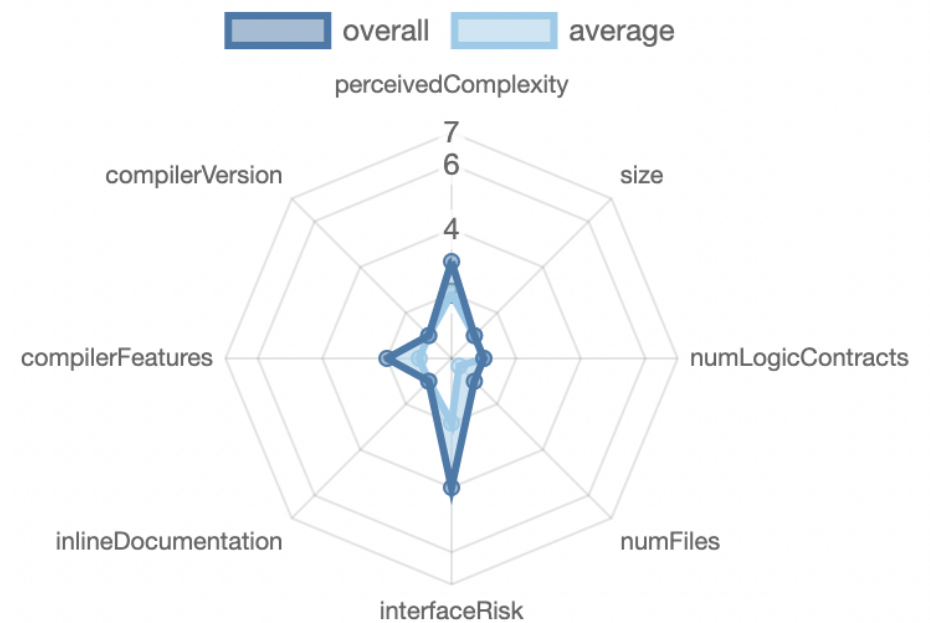
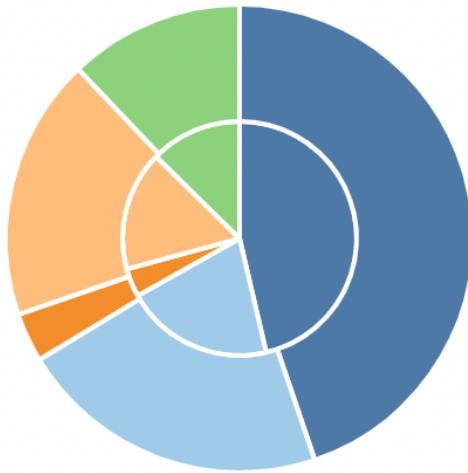


5.4 Inheritance Graph



5.5 Source Lines & Risk

source comment single block mixer
empty todo blockEmpty





5.6 Capabilities


Solidity Versions observed		 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
<div><div>^0.8.7</div><div>^0.8.0</div></div>			<div>yes</div>	<div>yes</div> <div>(1 asm blocks)</div>	<div></div>
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECRecover	 New/Create/Create2
<div></div>	<div></div>	<div></div>	<div>yes</div>	<div></div>	<div></div>

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			
26	6			
External	Internal	Private	Pure	View
18	16	1	4	10









StateVariables

Total	 Public
13	10

5.7 Source Unites in Scope

Source: <https://bitbucket.org/capsule-corp-labs/nft-drop-contract/src/master/contracts/>

Last commit: fbc7157f2e4df7c5fc99b8a260485fb70a6fc014

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/interfaces/ITwentyMint.sol		1	49	17	8	23	32	
	contracts/TwentyMint.sol	1		210	205	132	34	125	
	contracts/Utils/MerkleProof.sol	1		65	61	25	32	22	
	Totals	2	1	324	283	165	89	179	

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

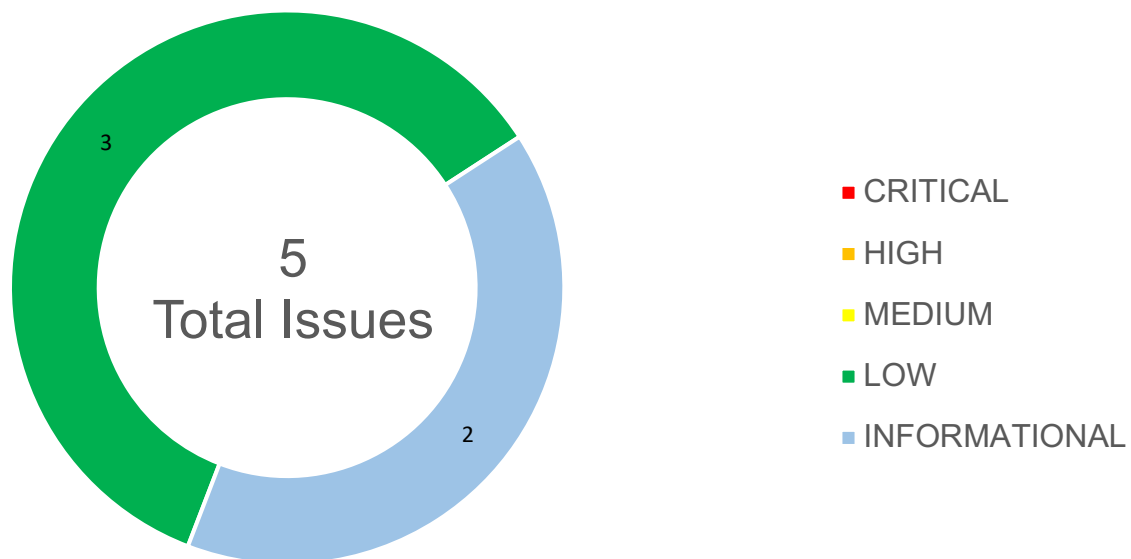
The Capsule Corp. Labs Team provided us with the files that needs to be tested. The scope of the audit is the 20 min NFT contract.

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 pre-mint / whitelist sale 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.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Spelling And Grammatical Errors	LOW	FIXED
6.2.2	Missing Zero Address Checks	LOW	FIXED
6.2.3	Renounce Ownership	LOW	FIXED
6.2.4	Floating Pragma Version Identified	INFORMATIONAL	FIXED
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 **3 Low issues** in the code of the smart contract.

6.2.1 Spelling And Grammatical Errors

Severity: LOW

Status: **FIXED**

Code: NA

File(s) affected: TwentyMint.sol

Update: <https://bitbucket.org/capsule-corp-labs/nft-drop-contract/commits/e2756b995e82304d506541f1854d2a2bcd8a7039>

Attack / Description	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.
Code	Line 132 – 143 (TwentyMint.sol) /**

	<pre> * @dev withdraw current balance: only owner can call * @param receipient address of receipient */ function withdraw(address payable receipient) external onlyOwner nonReentrant { require(receipient != address(0), "Invalid recipient address"); uint256 currentBalance = getBalance(); (bool success,) = receipient.call{ value: currentBalance }(""); require(success, "Failed to send ethers"); } </pre>
Result/Recommendation	<p>Better: recipient</p> <p>https://www.dictionary.com/browse/recipient</p>

6.2.2 Missing Zero Address Checks

Severity: LOW

Status: **FIXED**

Code: NA

File(s) affected: TwentyMint.sol

Update: <https://bitbucket.org/capsule-corp-labs/nft-drop-contract/commits/4e79b219ef99874b87c7712c27e6d803f3f9af29>

Attack / Description	In the current implementation, there are several addresses set without checking for the zero address. This can lead to unintended behaviour.
Code	<p>Line 203 – 205 (TwentyMint.sol)</p> <pre> function setRoyaltyInfo(address _receiver, uint96 _royaltyFeesInBips) public onlyOwner { _setDefaultRoyalty(_receiver, _royaltyFeesInBips); } </pre>

Result/Recommendation	It is highly recommended to check the addresses provided in the arguments for zero address (0).

6.2.3 Renounce Ownership

Severity: LOW

Status: **FIXED**

Code: CWE-283

File(s) affected: TwentyMint.sol

Update: <https://bitbucket.org/capsule-corp-labs/nft-drop-contract/commits/9d2ff3f6d384f95cc1d080199fdb0080cd5ca3b1>

Attack / Description	Typically, the contract's owner is the account that deploys the contract. As a result, the owner can perform certain privileged activities. The renounceOwnership function is used in smart contracts to renounce ownership. However, if the contract's ownership has never been transferred before renouncing it, it will never have an Owner, which may result in a denial of service.
Code	Line 12 (TwentyMint.sol) <pre>contract TwentyMint is ERC721Enumerable, ERC2981, ReentrancyGuard, Ownable {</pre>
Result/Recommendation	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 will require two or more users to sign the transaction. Alternatively, the Renounce Ownership functionality can be disabled by overriding it. <pre>/** * @dev function "renounceOwnership()" prevents ownership renounce */ function renounceOwnership() public override onlyOwner { }</pre>

INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **2 Informational issues** in the code of the smart contract.

6.2.4 Floating Pragma Version Identified

Severity: INFORMATIONAL

Status: **FIXED**

Code: SWC-103

File(s) affected: ALL

Update: <https://bitbucket.org/capsule-corp-labs/nft-drop-contract/commits/541d5d38cbf523b138f2276dd5af56a7d523a7e1>

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 <code>pragma solidity ^0.8.7;</code>
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 0.8.7

6.2.5 Storing Data Via baseURI

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

Code: NA

File(s) affected: TwentyMint.sol

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 158 - 160 (TwentyMint.sol) <pre> function setBaseURI(string memory _baseURI) external onlyOwner { baseURI = _baseURI; } </pre>
Result/Recommendation	<p>We recommend using IPFS and pinning services to make the metadata behind the baseURI permanently stored.</p> <p>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.</p> <p>Check more information here: https://docs.ipfs.io/concepts/persistence/#persistence-versus-permanence</p> <p>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.</p> <p>Please investigate SVG generated on-chain visuals and on-chain stored metadata, for persistent storage.</p>

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 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

6.4.2 Owner cannot mint any new tokens after pre-mint / whitelist sale and public sale minting was done


Status: tested and verified 

6.4.3 Owner cannot burn or lock user NFTs

Status: tested and verified 


There is no burn or lock function

6.4.4 Owner cannot pause the contract

Status: tested and verified 

There is no pause function

6.4.5 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 of the smart contract. During the audit, no critical, only 3 Low and 2 Informational issues have been found, after the manual and automated security testing. We advise the Capsule Corp Labs. team to implement the recommendations to further enhance the code's security and readability.

8. Deployed Smart Contract

VERIFIED

<https://etherscan.io/address/0xB003ce92F3b2A8F3dd99207C351eAf05BC605262#code>



9. About the Auditor

Chainsulting is a professional software development firm based in Germany that provides comprehensive distributed ledger technology (DLT) 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 secured 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.

The blockchain security provider brings the highest security standards to crypto and blockchain platforms, helping to foster growth and transparency within the fast-growing ecosystem.

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.