



**NFT TiX**

**Marketplace**

**SMART CONTRACT AUDIT**

**16.08.2022**

**Made in Germany by Chainsulting.de**



## Table of contents

1. Disclaimer.....	3
2. About the Project and Company .....	4
2.1 Project Overview.....	5
3. Vulnerability & Risk Level .....	6
4. Auditing Strategy and Techniques Applied.....	7
4.1 Methodology .....	7
5. Metrics .....	8
5.1 Tested Contract Files .....	8
5.2 Used Code from other Frameworks/Smart Contracts .....	9
5.3 CallGraph .....	10
5.4 Inheritance Graph .....	11
5.5 Source Lines & Risk .....	12
5.6 Capabilities .....	13
5.7 Source Unites in Scope .....	14
6. Scope of Work.....	15
6.1 Findings Overview .....	16
6.2 Manual and Automated Vulnerability Test.....	17
6.2.1 Centralization Risk .....	17
6.2.2 Missing Natspec Documentation .....	18
6.3 SWC Attacks .....	19
6.4. Verify Claims .....	22
7. Executive Summary.....	23

8. Deployed Smart Contract .....	23
9. About the Auditor .....	24

## 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 NFT-TiX LLC. 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 (03.08.2022)	Layout
0.4 (08.08.2022)	Automated Security Testing Manual Security Testing
0.5 (10.08.2022)	Verify Claims and Test Deployment
0.6 (12.08.2022)	Testing SWC Checks
0.9 (14.08.2022)	Summary and Recommendation
1.0 (14.08.2022)	Final document
1.1 (TBA)	Deployed contracts

## 2. About the Project and Company



**Company address:**

NFT-TiX LLC  
457 Alpine Ter, Ridgewood  
New Jersey 07450  
United States of America

**Website:** <https://nft-tix.com>

**Twitter:** [https://twitter.com/nft\\_tix](https://twitter.com/nft_tix)

**Instagram:** [https://www.instagram.com/nft\\_tix](https://www.instagram.com/nft_tix)

**Blog:** <https://nft-tix.com/blog>

**E-Mail:** [info@nft-tix.com](mailto:info@nft-tix.com)

## 2.1 Project Overview

NFT-TiX Marketplace is breaking down the barrier between ticket buyers and ticket sellers. For the first time ever, there will be no place for the intermediary processors. NFT-TiX Marketplace does not require any central authority. There are no executives peering down into the process and reaching in their hands to take a cut. This means that ticket buyers and ticket sellers can interact with each other from anywhere in the world, without relying on any third parties. Combining blockchain technology with the ticketing experience, we have put together a straightforward but innovative process. Event Organizers generate their own tickets, as NFTs, running smart contracts in order to do so. Buyers then run smart contracts of their own, to process payment for the tickets and take ownership. This protocol will be familiar enough that blockchain, cryptocurrency, and NFT enthusiasts will see its value immediately.

We are making it so simple, in fact, that even those who have never touched cryptocurrency at all will be able to jump in and use it. Our vision is of a ticketing system that feels personal – and that runs according to democratic principles, championing access and efficiency above all else. Whereas the ticket monopolies of today have forced sellers to raise prices and forced buyers to pay those prices, NFT-TiX Marketplace allows the sellers themselves to decide what to charge. This is a power that ticket companies previously kept from the sellers, in order to continue to take advantage of the buyers. Long-term, NFT-TiX Marketplace is going to reshape this industry. Sports, concerts, magic shows, circuses, car shows, special parties, and much, much more: our technology is widely applicable, to any event that requires a ticket.

There is no gating to prevent anyone from selling tickets – unless the sellers decide to restrict or limit sales in some way, by setting those details in their smart contracts. This is a protocol that will work better for everyone. Sellers who use NFT-TiX Marketplace can expect to earn more from their tickets, to feel more in control of the terms of their sales, and to save themselves time and resources that they would have expended on dealing with the ticketing companies. Buyers who use NFT-TiX Marketplace will, on the other hand, benefit from lower prices, a smoother buying experience, and greater freedom to do with their tickets as they please. In the meantime, the bulky, inefficient, monopolistic ticketing platforms are going to become relics of the past. We believe that now is the time to push them out, to redirect this space so that it is working for people, not corporations.

### 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

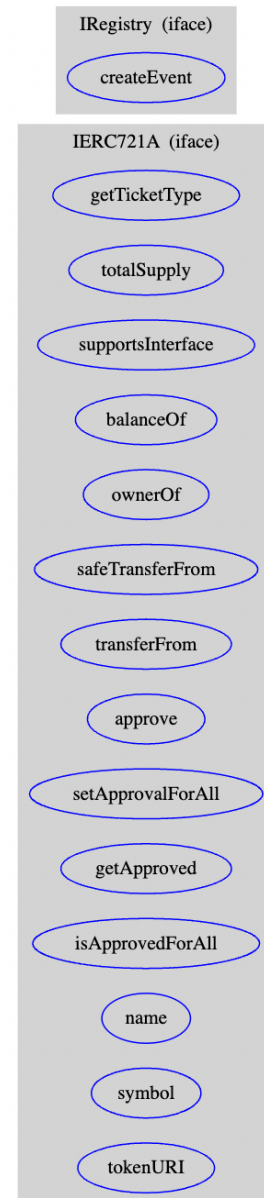
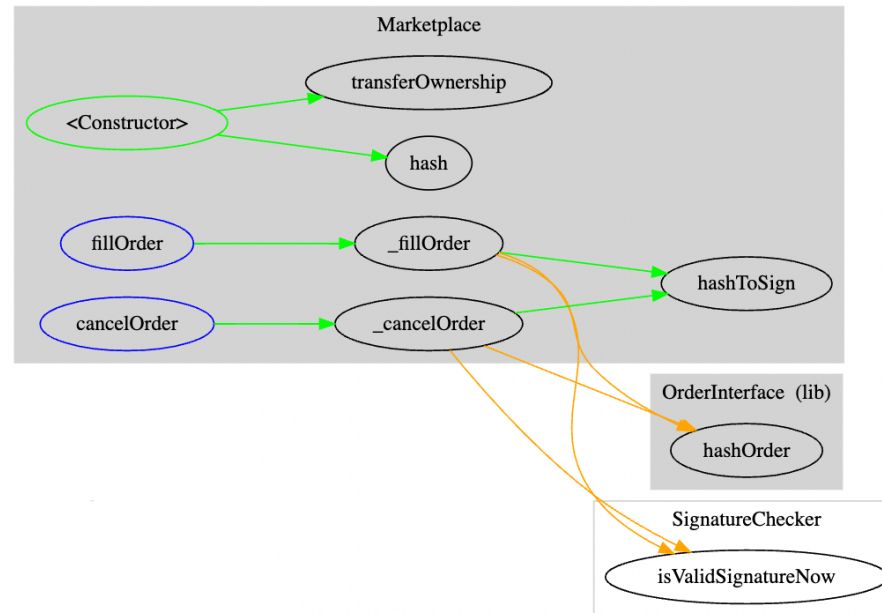
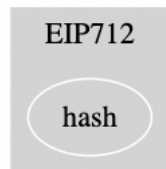
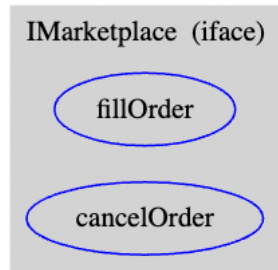
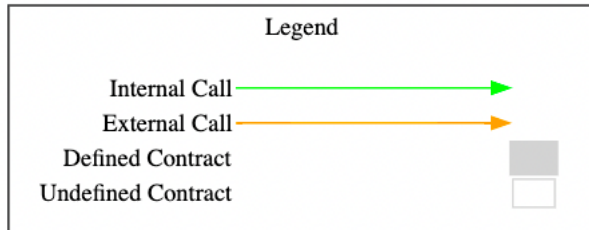
File	Fingerprint (MD5)
./interfaces/IMarketplace.sol	2e89fa5c8e6e445c65d084ffe7e235d3
./interfaces/IERC721A.sol	94c68365be36db5f69172b16c3bc0c98
./interfaces/IRegistry.sol	509863ab4b22b2e24d662fa0528007d4
./lib/OrderInterface.sol	3c33d21b805f3f5c6be4b6cb69680a9d
./Marketplace.sol	384e0dbaeaae83b3b3118bd14fcd1678
./extensions/EIP712.sol	a37d94aa1740238305995344609f6958



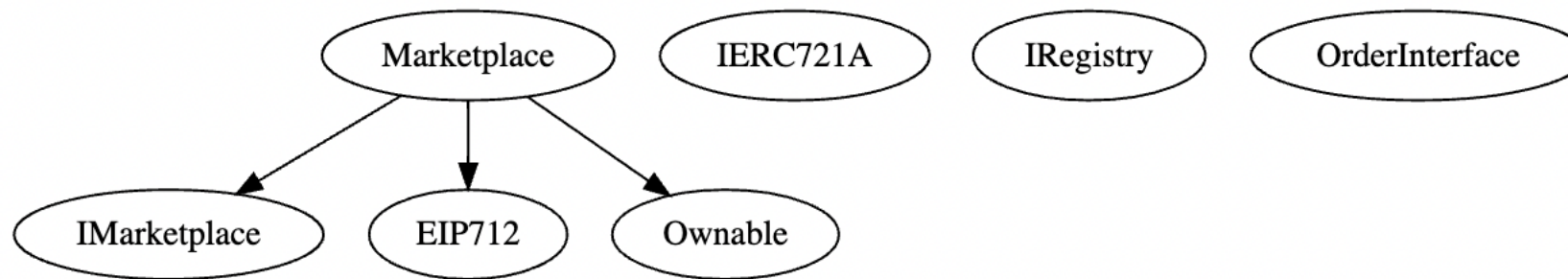
## 5.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
@openzeppelin/contracts/access/Ownable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/contracts/access/Ownable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/contracts/access/Ownable.sol</a>
@openzeppelin/contracts/utils/cryptography/SignatureChecker.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/contracts/utils/cryptography/SignatureChecker.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/contracts/utils/cryptography/SignatureChecker.sol</a>
erc721a/contracts/IERC721A.sol	<a href="https://github.com/chiru-labs/ERC721A">https://github.com/chiru-labs/ERC721A</a>

## 5.3 CallGraph

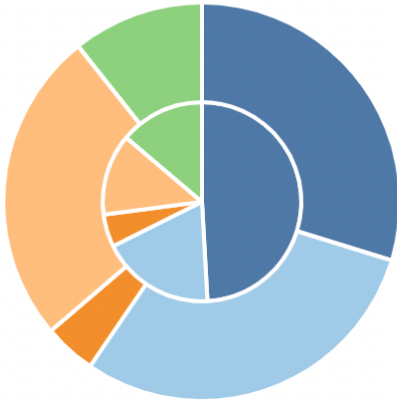


## 5.4 Inheritance Graph

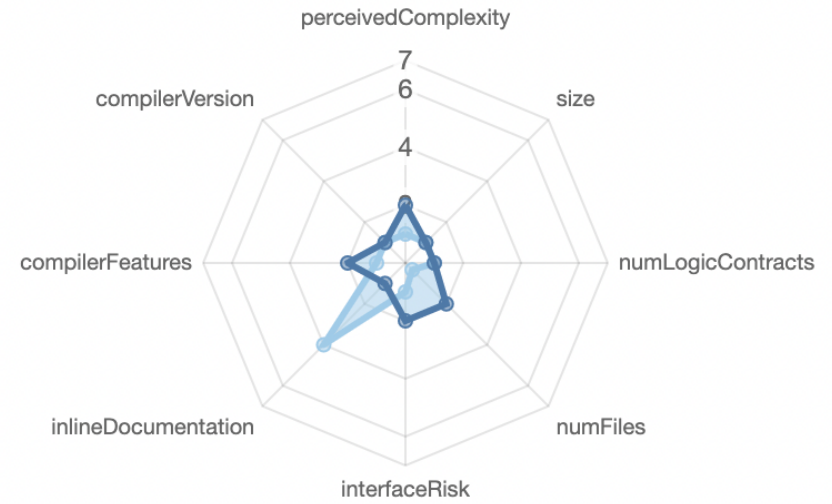


## 5.5 Source Lines & Risk





source comment single block mixed  
empty todo blockEmpty



overall average





## 5.6 Capabilities


Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts	
<div>0.8.11</div> <div>^0.8.4</div>					
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECRrecover	 New/Create/Create2
			<div>yes</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				
20	0				
External	Internal	Private	Pure	View	
20	18	0	2	11	












### StateVariables

Total	 Public
8	3

## 5.7 Source Unites in Scope

Source: <https://github.com/nfttix/contracts>

Last commit: 32b60c442144ab205ed2e1577c5b3030ab857dd4

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/interfaces/IMarketplace.sol	_____	1	16	9	4	2	5	_____
	contracts/interfaces/IERC721A.sol	_____	1	265	121	54	177	31	_____
	contracts/interfaces/IRegistry.sol	_____	1	22	12	7	2	3	_____
	contracts/lib/OrderInterface.sol	1	_____	37	37	31	2	6	
	contracts/Marketplace.sol	1	_____	74	66	47	4	39	
	contracts/extensions/EIP712.sol	1	_____	32	32	24	2	11	
	<b>Totals</b>	<b>3</b>	<b>3</b>	<b>446</b>	<b>277</b>	<b>167</b>	<b>189</b>	<b>95</b>	

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

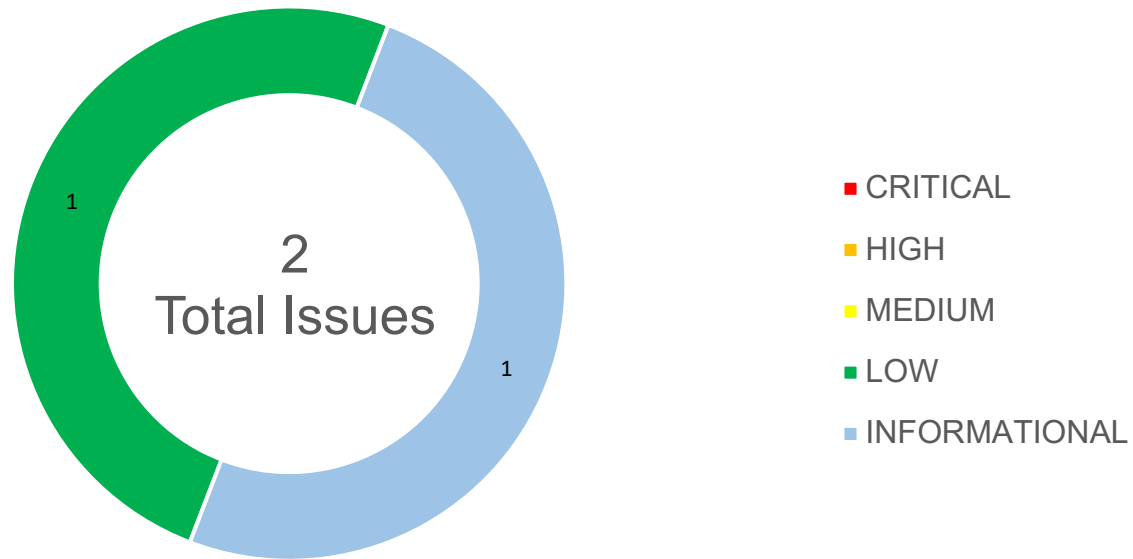
The NFT TiX Team provided us with the files that needs to be tested. The scope of the audit is the Marketplace contract.

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

- Signatures are done a secure way and according to the newest standards
- 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	Centralization Risk	LOW	FIXED
6.2.2	Missing Natspec Documentation	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 Centralization Risk

Severity: LOW

Status: FIXED

Code: NA

File(s) affected: Marketplace.sol

Update: The Team put all required security measurements in place, to reduce the centralization risk by a minimum.

<b>Attack / Description</b>	One of the features of NFT TiX is the gasless listing of NFTs. That requires the user to sign a message, to list the ticket on the marketplace. When another user buys the ticket off the marketplace (filling the order) they are then paying the fee for submitting this order on-chain. The signatures are stored in a centralized database, with a logic to match the orders. This backend can lead to a centralized point of failure and can be target to hacker attacks.
-----------------------------	--

<b>Code</b>	Line 34 – 40 (Marketplace.sol) <pre> function fillOrder(     OrderInterface.Order memory order,     bytes memory signature,     address buyer ) external onlyOwner {     _fillOrder(order, signature, buyer); } </pre>
<b>Result/Recommendation</b>	If the Owner wallet/private key gets into the wrong hands, caused by a leak or hack, then it's possible to harm the project. We recommend protecting the Owner wallet with a multi-signature structure, such as gnosis safe or similar. Also, we are highly recommending the team to hardening the backend and enable firewall/ ddos protection.

## INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **1 Informational issue** in the code of the smart contract.

### 6.2.2 Missing Natspec Documentation

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

Code: NA

File(s) affected: ALL

<b>Attack / Description</b>	Solidity contracts can use a special form of comments to provide rich documentation for function, return variables, and more. This special form is named Ethereum Natural Language Specification Format(NatSpec).
<b>Code</b>	//

<b>Result/Recommendation</b>	It is recommended to include natspec documentation and follow the doxygen style including @author, @title, @notice, @dev, @param, @return and make it easier to review and understand your smart contract.
------------------------------	--

## 6.3 SWC Attacks

ID	Title	Relationships	Test Result
<a href="#">SWC-131</a>	Presence of unused variables	<a href="#">CWE-1164: Irrelevant Code</a>	✓
<a href="#">SWC-130</a>	Right-To-Left-Override control character (U+202E)	<a href="#">CWE-451: User Interface (UI) Misrepresentation of Critical Information</a>	✓
<a href="#">SWC-129</a>	Typographical Error	<a href="#">CWE-480: Use of Incorrect Operator</a>	✓
<a href="#">SWC-128</a>	DoS With Block Gas Limit	<a href="#">CWE-400: Uncontrolled Resource Consumption</a>	✓
<a href="#">SWC-127</a>	Arbitrary Jump with Function Type Variable	<a href="#">CWE-695: Use of Low-Level Functionality</a>	✓
<a href="#">SWC-125</a>	Incorrect Inheritance Order	<a href="#">CWE-696: Incorrect Behavior Order</a>	✓

ID	Title	Relationships	Test Result
<a href="#">SWC-124</a>	Write to Arbitrary Storage Location	<a href="#">CWE-123: Write-what-where Condition</a>	✓
<a href="#">SWC-123</a>	Requirement Violation	<a href="#">CWE-573: Improper Following of Specification by Caller</a>	✓
<a href="#">SWC-122</a>	Lack of Proper Signature Verification	<a href="#">CWE-345: Insufficient Verification of Data Authenticity</a>	✓
<a href="#">SWC-121</a>	Missing Protection against Signature Replay Attacks	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	✓
<a href="#">SWC-120</a>	Weak Sources of Randomness from Chain Attributes	<a href="#">CWE-330: Use of Insufficiently Random Values</a>	✓
<a href="#">SWC-119</a>	Shadowing State Variables	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓
<a href="#">SWC-118</a>	Incorrect Constructor Name	<a href="#">CWE-665: Improper Initialization</a>	✓
<a href="#">SWC-117</a>	Signature Malleability	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	✓
<a href="#">SWC-116</a>	Timestamp Dependence	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	✓

ID	Title	Relationships	Test Result
<a href="#">SWC-115</a>	Authorization through tx.origin	<a href="#">CWE-477: Use of Obsolete Function</a>	✓
<a href="#">SWC-114</a>	Transaction Order Dependence	<a href="#">CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</a>	✓
<a href="#">SWC-113</a>	DoS with Failed Call	<a href="#">CWE-703: Improper Check or Handling of Exceptional Conditions</a>	✓
<a href="#">SWC-112</a>	Delegatecall to Untrusted Callee	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	✓
<a href="#">SWC-111</a>	Use of Deprecated Solidity Functions	<a href="#">CWE-477: Use of Obsolete Function</a>	✓
<a href="#">SWC-110</a>	Assert Violation	<a href="#">CWE-670: Always-Incorrect Control Flow Implementation</a>	✓
<a href="#">SWC-109</a>	Uninitialized Storage Pointer	<a href="#">CWE-824: Access of Uninitialized Pointer</a>	✓
<a href="#">SWC-108</a>	State Variable Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓
<a href="#">SWC-107</a>	Reentrancy	<a href="#">CWE-841: Improper Enforcement of Behavioral Workflow</a>	✓
<a href="#">SWC-106</a>	Unprotected SELFDESTRUCT Instruction	<a href="#">CWE-284: Improper Access Control</a>	✓

ID	Title	Relationships	Test Result
<a href="#">SWC-105</a>	Unprotected Ether Withdrawal	<a href="#">CWE-284: Improper Access Control</a>	✓
<a href="#">SWC-104</a>	Unchecked Call Return Value	<a href="#">CWE-252: Unchecked Return Value</a>	✓
<a href="#">SWC-103</a>	Floating Pragma	<a href="#">CWE-664: Improper Control of a Resource Through its Lifetime</a>	✓
<a href="#">SWC-102</a>	Outdated Compiler Version	<a href="#">CWE-937: Using Components with Known Vulnerabilities</a>	✓
<a href="#">SWC-101</a>	Integer Overflow and Underflow	<a href="#">CWE-682: Incorrect Calculation</a>	✓
<a href="#">SWC-100</a>	Function Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓

## 6.4. Verify Claims

6.4.1 Signatures are done a secure way and according to the newest standards

**Status:** tested and verified ✓

6.4.2 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, no high, no medium, one low and one Informational issues have been found, after the manual and automated security testing. We advise the NFT-Tix team to implement the recommendations to further enhance the code's security and readability.

## 8. Deployed Smart Contract

PENDING

## 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 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.

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.