



NFT-TiX

Event Factory

SMART CONTRACT AUDIT

06.07.2022

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	11
5.4 Inheritance Graph	13
5.5 Source Lines & Risk	14
5.6 Capabilities	15
5.7 Source Unites in Scope	16
6. Scope of Work.....	17
6.1 Findings Overview	18
6.2 Manual and Automated Vulnerability Test.....	19
6.2.1 Floating Pragma Version Identified	19
6.2.2 Storing Data Via baseURI.....	20
6.2.3 Missing Natspec Documentation	21
6.3 SWC Attacks	22
6.4. Verify Claims	26

7. Executive Summary.....	26
8. Deployed Smart Contract	26
9. About the Auditor	27

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 (18.04.2022)	Layout
0.4 (20.04.2022)	Automated Security Testing Manual Security Testing
0.5 (24.04.2022)	Verify Claims and Test Deployment
0.6 (25.04.2022)	Testing SWC Checks
0.9 (26.04.2022)	Summary and Recommendation
1.0 (29.04.2022)	Final document
1.1 (04.05.2022)	Re-check
1.2 (06.07.2022)	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

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

E-Mail: 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

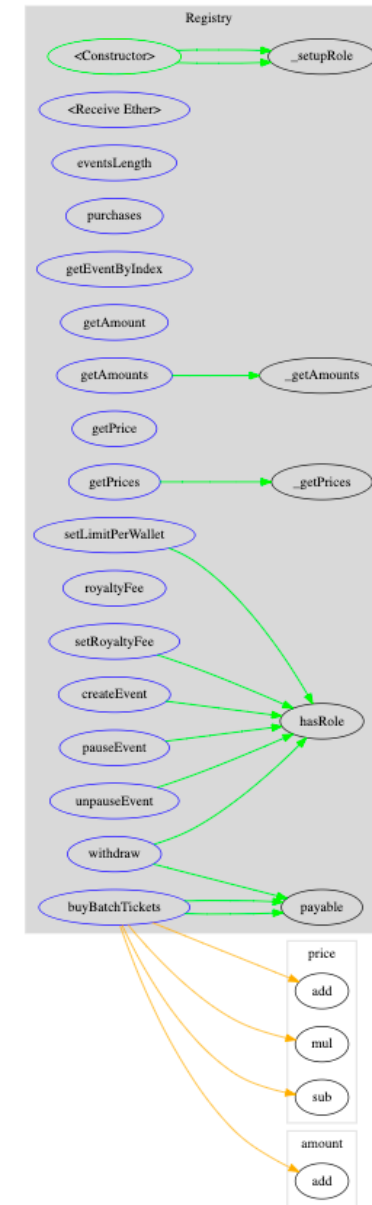
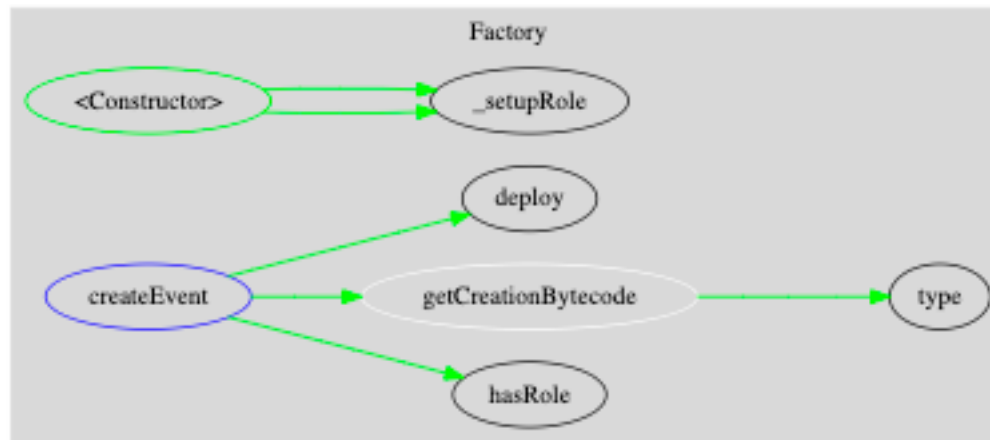
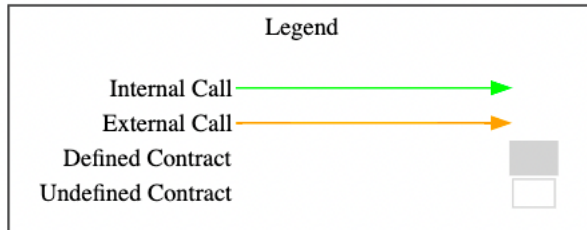
File	Fingerprint (MD5)
./Event.sol	54fb2ad0656cb5d8e8d87c4e6733b4f4
./Factory.sol	e0c46653a988649625e5e0f65f05f244
./Registry.sol	9426325228f46c81716ebb9690d25bf2
./interfaces/IERC2309.sol	dc9d5b6d6915c96855d70409182612c7
./interfaces/IEvent.sol	24a53d4ea596cf5da7408bb0e42cf137
./interfaces/IFactory.sol	354d17f8400a59decefc2e5ceae65426
./interfaces/IRegistry.sol	944ffb083ea27b92ba424975299efbfe

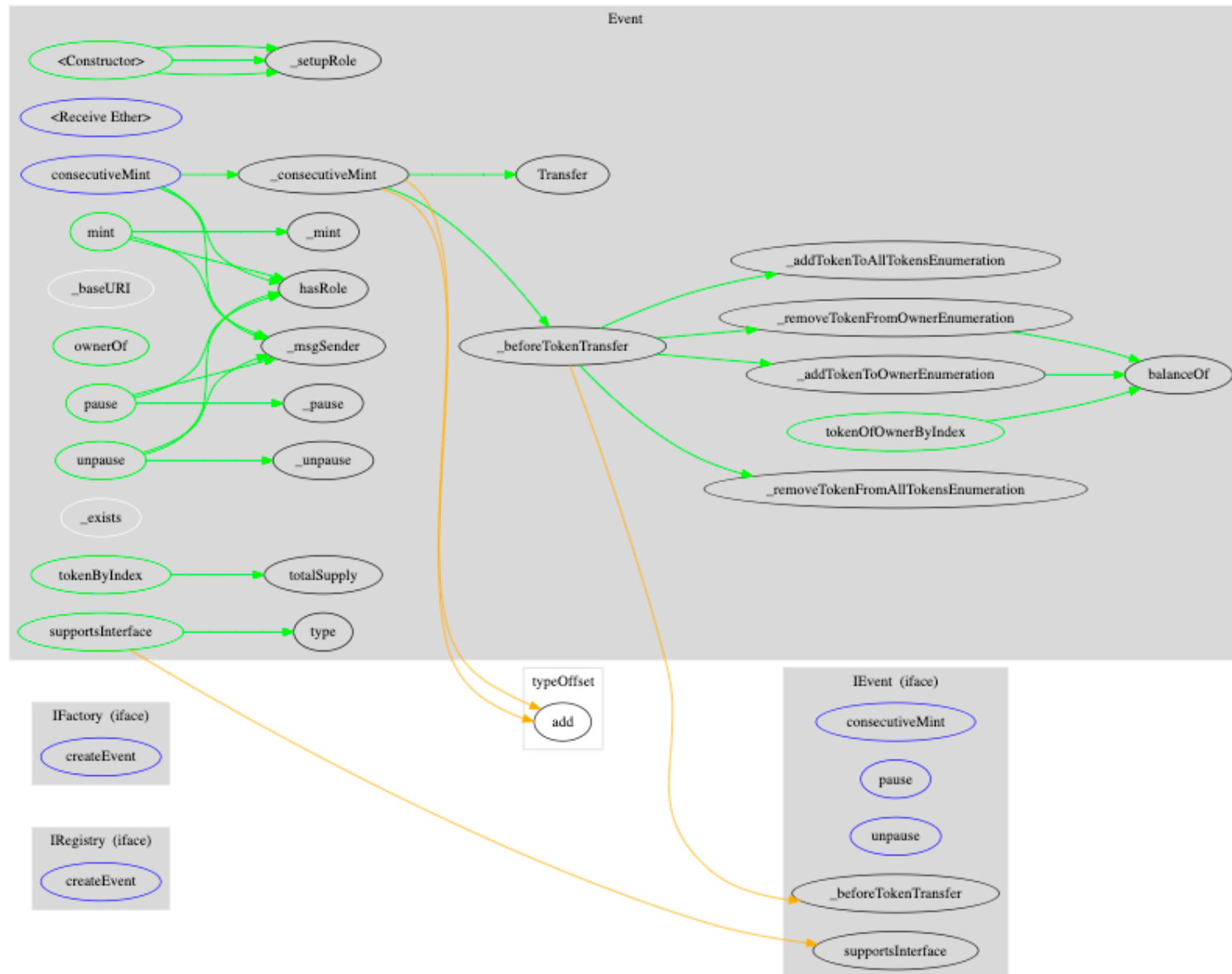
5.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
@openzeppelin/contracts/access/AccessControl.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/access/AccessControl.sol
@openzeppelin/contracts/access/AccessControlEnumerable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/access/AccessControlEnumerable.sol
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/access/Ownable.sol
@openzeppelin/contracts/token/ERC721/ERC721.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/token/ERC721/ERC721.sol
@openzeppelin/contracts/token/ERC721/extensions/ERC721Burnable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/token/ERC721/extensions/ERC721Burnable.sol
@openzeppelin/contracts/token/ERC721/extensions/ERC721Pausable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/token/ERC721/extensions/ERC721Pausable.sol
@openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/token/ERC721/extensions/IERC721Enumerable.sol
@openzeppelin/contracts/utils/Context.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/utils/Context.sol

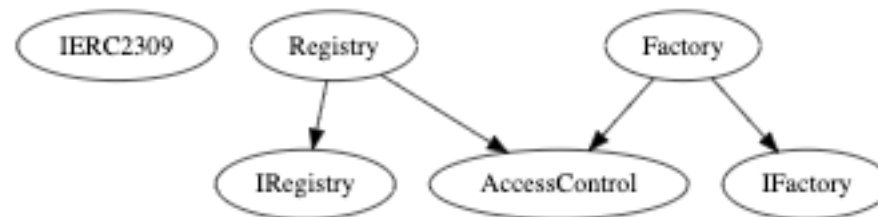
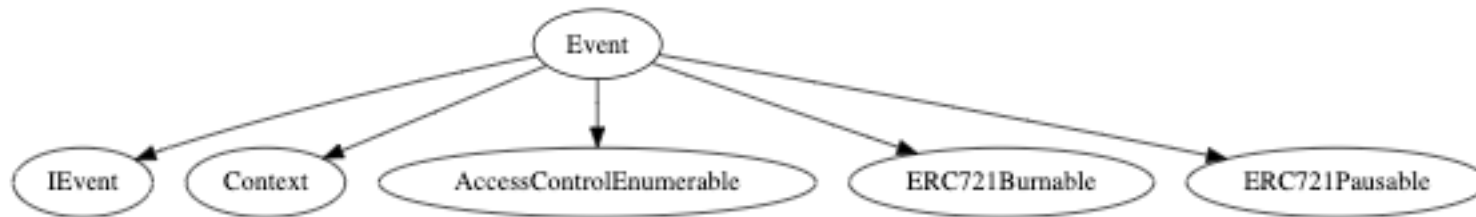
Dependency / Import Path	Source
@openzeppelin/contracts/utils/Counters.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/contracts/Counters.sol
@openzeppelin/contracts/contracts/math/SafeMath.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.2/contracts/math/SafeMath.sol

5.3 CallGraph



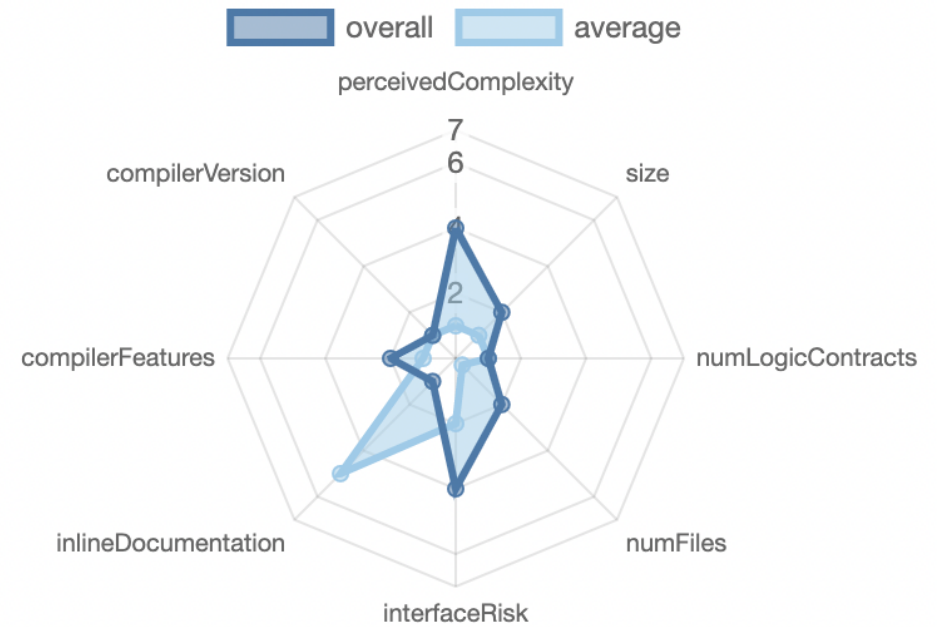
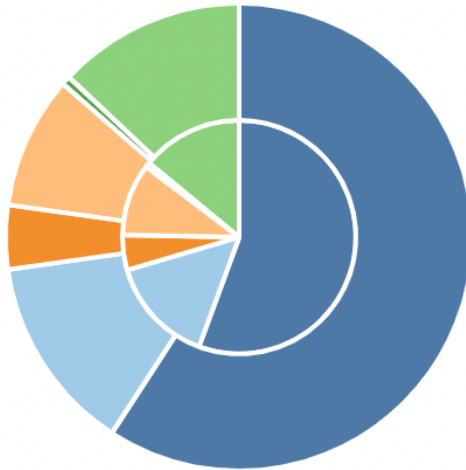


5.4 Inheritance Graph













5.5 Source Lines & Risk

source comment single block mixed
empty todo blockEmpty





5.6 Capabilities

Solidity Versions observed		 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
<div>>0.8.0</div> <div>^0.8.0</div>			<div>yes</div>	<div>yes</div> <div>(1 asm blocks)</div>	
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 Ecrecover	 New/Create/Create2
<div>yes</div>			<div>yes</div>		<div>yes</div> <div>→ AssemblyCall:Name:create2</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			
33	3			
External	Internal	Private	Pure	View
24	34	4	1	18






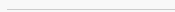




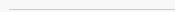
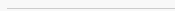




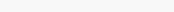





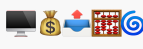
StateVariables

Total	Public
20	5

5.7 Source Unites in Scope

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

Last commit: f5b683b5d33b69c6b20e5793b016cf51a2a323db2

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/interfaces/IERC2309.sol		1	6	6	4	1	1	
	contracts/interfaces/IRegistry.sol		1	13	5	3	1	3	
	contracts/interfaces/IFactory.sol		1	11	5	3	1	3	
	contracts/interfaces/IEvent.sol		1	13	5	3	1	7	
	contracts/Registry.sol	1		185	174	141	3	157	
	contracts/Event.sol	1		278	266	147	81	128	
	contracts/Factory.sol	1		51	41	32	1	45	
	Totals	3	4	557	502	333	89	344	

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

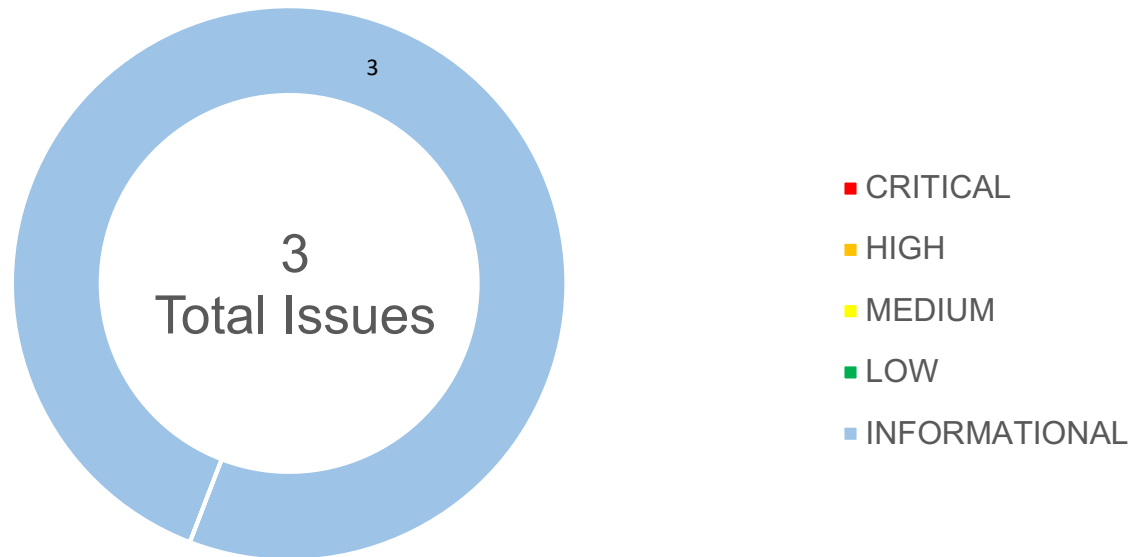
The NFT-Tix Team provided us with the files that needs to be tested. The scope of the audit are the Event Factory contracts.

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

- The contract is using and ERC standard for NFTs
- 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	Floating Pragma Version Identified	INFORMATIONAL	ACKNOWLEDGED
6.2.2	Storing Data Via baseURI	INFORMATIONAL	ACKNOWLEDGED
6.2.3	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 **0 Low issues** in the code of the smart contract.

INFORMATIONAL ISSUES

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

6.2.1 Floating Pragma Version Identified

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

Code: SWC-103

File(s) affected: ALL

Update: We will specify a fixed version in one of our next updates. Our updates run in batches for practical reasons, so this will be applied in the near future. We are fine with this at the moment.

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 <pre>pragma solidity >0.8.0; pragma solidity ^0.8.0;</pre>
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

6.2.2 Storing Data Via baseURI

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

Code: NA

File(s) affected: Event.sol

Update: We have a proxy function on our backend that is handling all the files and uploading them to IPFS. Owner/creator in this case is us and it is required for us to be able to have this control. This is working as intended.

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 102 - 104 (Event.sol) <pre>function _baseURI() internal view virtual override returns (string memory) { return _baseTokenURI; }</pre>
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.

	<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>
--	---

6.2.3 Missing Natspec Documentation

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

Code: NA

File(s) affected: Registry.sol, Factory.sol

Update: This is also something we will roll out in the future as part of documenting the final version.

Attack / Description	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).
Code	//
Result/Recommendation	

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
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	✓


ID	Title	Relationships	Test Result
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	✓
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	✓

ID	Title	Relationships	Test Result
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	✓
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	✓

ID	Title	Relationships	Test Result
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	✗
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 an correctly implemented ERC-721 standard

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, only three 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

VERIFIED

<https://etherscan.io/address/0xed73c43c27eef0a7c167d9597a1b4203dbafce95#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 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.