IEX

Dispatch Core

SMART CONTRACT AUDIT

18.08.2022

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 Digital Asset Communications 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 (27.05.2022)	Layout
0.4 (29.05.2022)	Automated Security Testing Manual Security Testing
0.5 (01.06.2022)	Verify Claims and Test Deployment
0.6 (05.06.2022)	Testing SWC Checks
0.9 (05.06.2022)	Summary and Recommendation
1.0 (06.06.2022)	Final document
1.1 (18.08.2022)	Added deployed contract

2. About the Project and Company

Company address:

Digital Asset Communications LLC
c/o Dispatch
3 World Trade Center, 58th Floor New York
NY 10007, USA

Website: https://dispatch.xyz

Twitter: https://twitter.com/dispatchxyz

Email: gm@dispatch.xyz

2.1 Project Overview

Dispatch is a decentralized community engagement platform, they bring on-chain messages directly to the investors via NFT technology. Projects are able to send messages to self-custody wallets and centralized platforms. They are able to understand, segment and target the audience in one place. There are different use-cases, for example DAOs are able to keep their members informed on governance proposals and event invites.

Dispatch is built by the team behind Investors Exchange (IEX), which is a stock exchange based in the United States. It was founded in 2012 in order to mitigate the effects of high-frequency trading. IEX was launched as a national securities exchange in September 2016. On October 24, 2017, IEX received regulatory approval from the SEC to list companies.

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

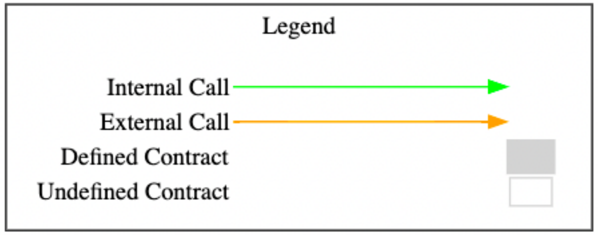
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability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

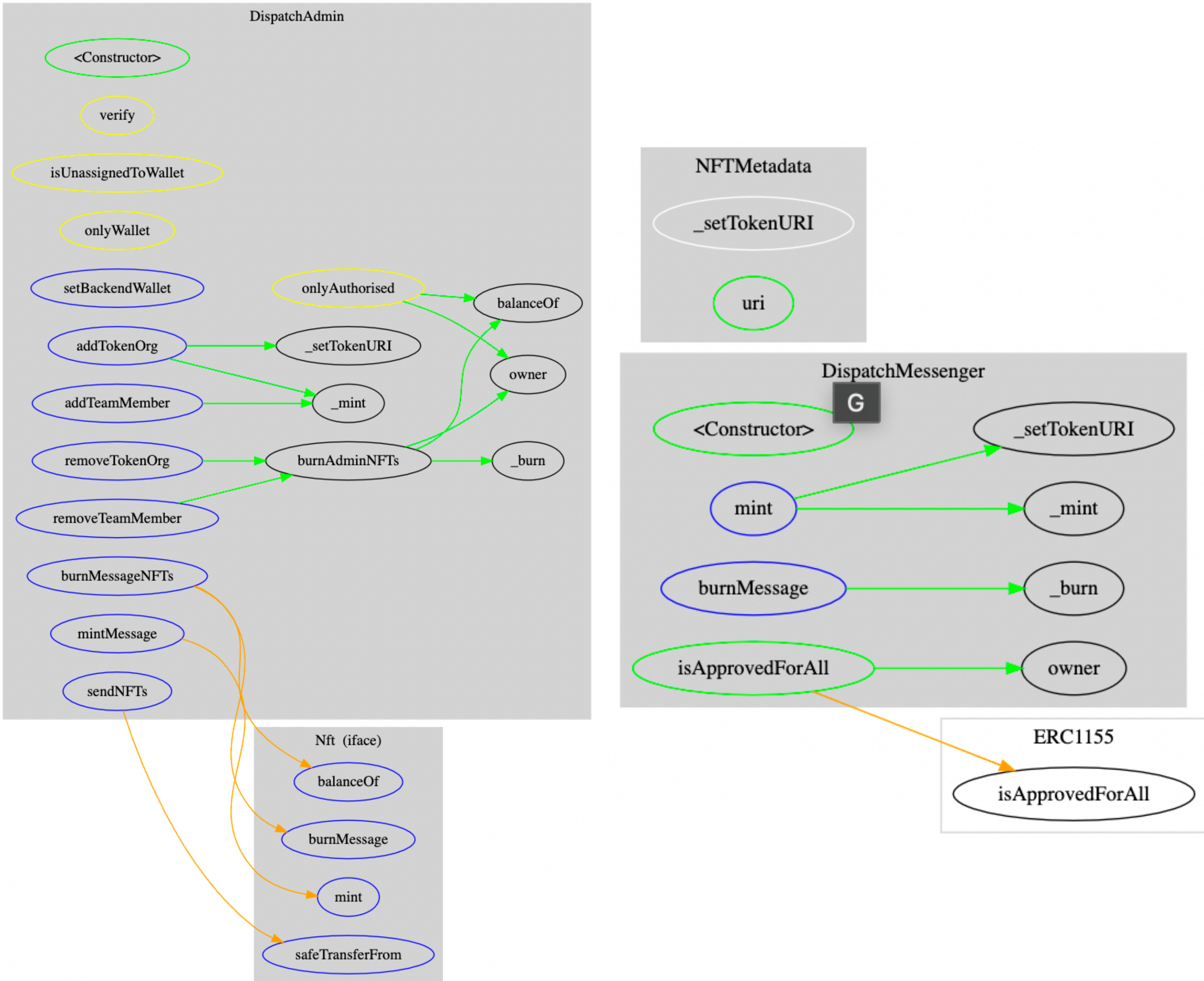
File	Fingerprint (MD5)
./contracts/Admin.sol	2a855203b42d60b643b479e0b859f514
./contracts/Messenger.sol	8cef8aec2c306d27e70f4ffcf3ce07b8
./contracts/NFTMetadata.sol	fc3618c443bccffb342a5d501402db8e
./contracts/interfaces/nft-interface.sol	36a4138ebbd07a474057a2db74512519

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

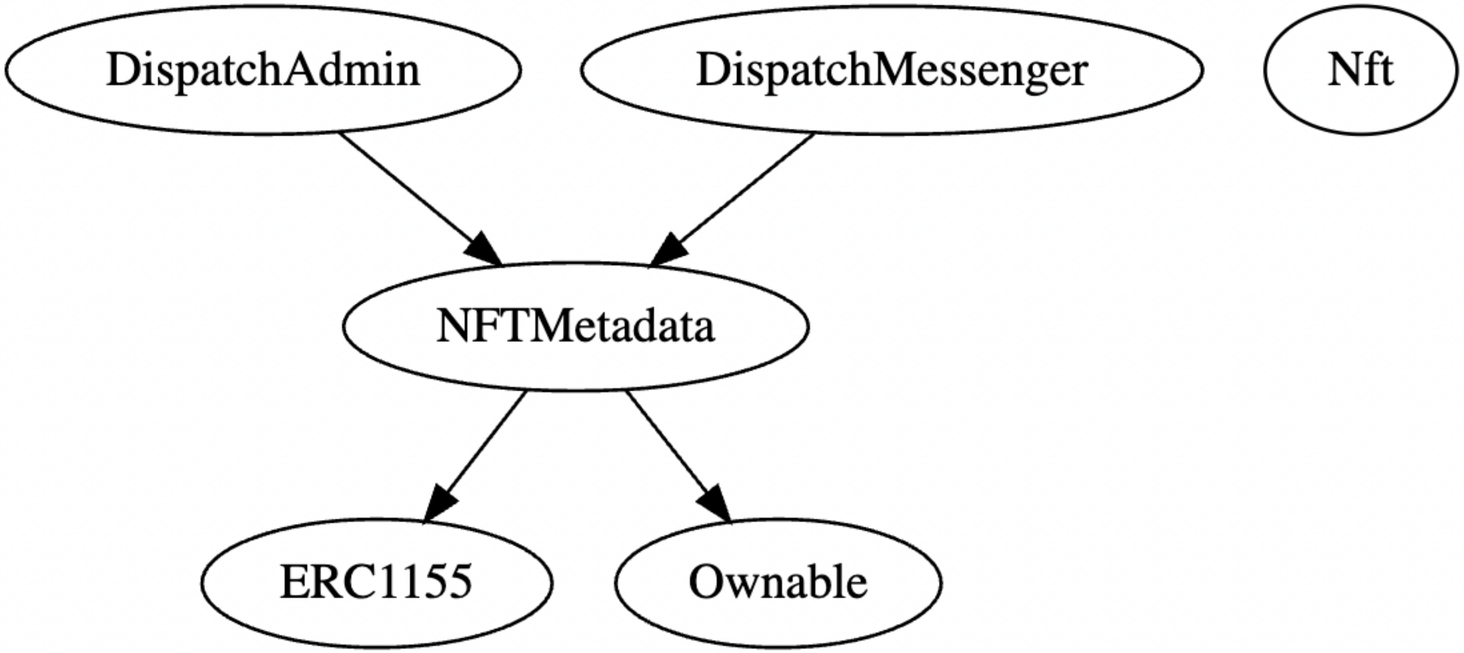
Dependency / Import Path	Source
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.2/contracts/access/Ownable.sol
@openzeppelin/contracts/token/ERC1155/ERC1155.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.2/contracts/token/ERC1155/ERC1155.sol

5.3 CallGraph

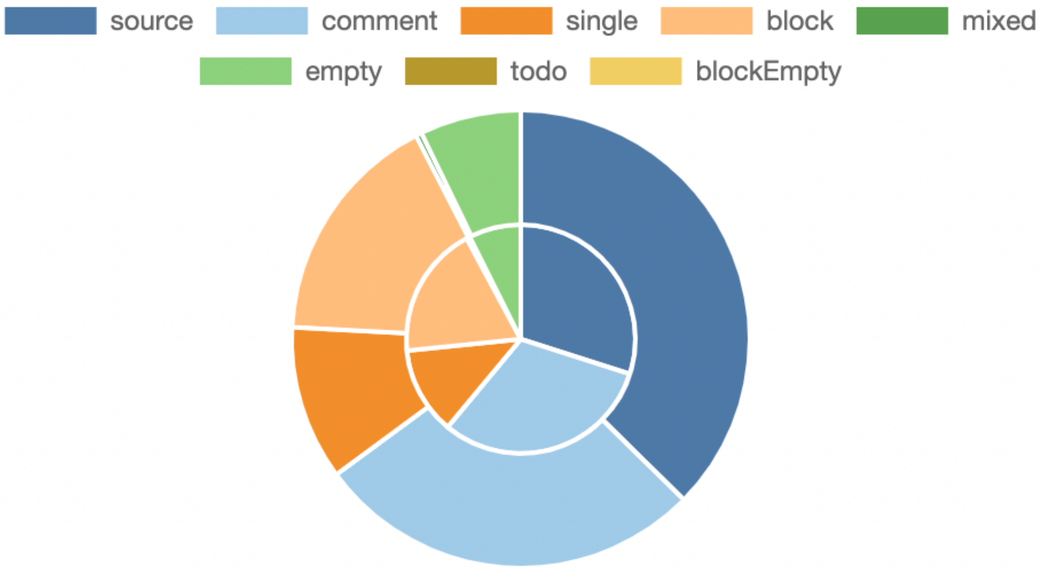


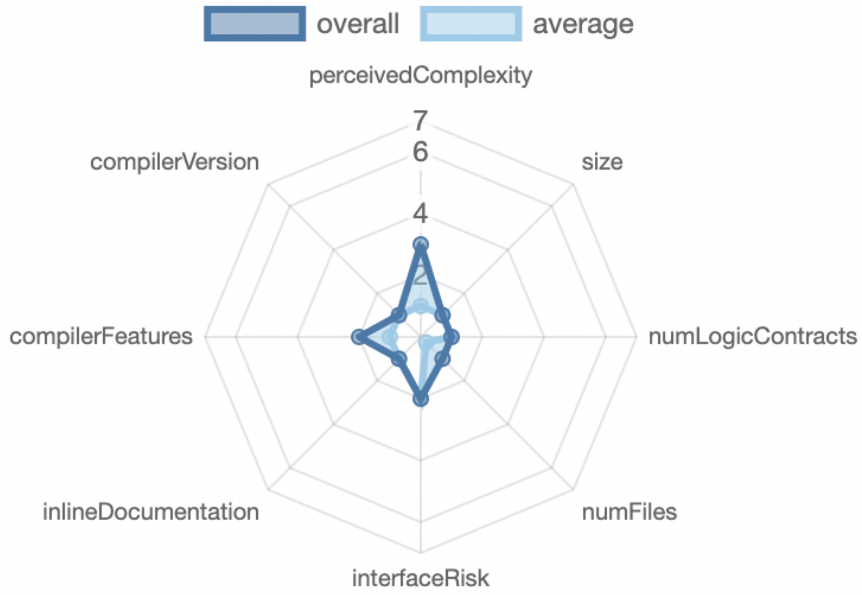


5.4 Inheritance Graph



5.5 Source Lines & Risk





5.6 Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.8.7` `^0.8.13`		`yes`		

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

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
17	1

External	Internal	Private	Pure	View
14	18	0	0	2

StateVariables

Total	🌐Public
7	3

5.7 Source Unites in Scope

Source: https://github.com/iex-xyz/iex-ethereum
Commit: 693260443c5760f3e50814f130c3826a2fc0e140

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	contracts/Admin.sol	1		219	193	86	85	85	💰
🔍	contracts/interfaces/nft-interface.sol		1	26	5	3	1	9	
📝	contracts/Messenger.sol	1		59	51	22	26	19	
🎨	contracts/NFTMetadata.sol	1		37	37	15	19	11	
📝🔍🎨	**Totals**	**3**	**1**	**341**	**286**	**126**	**131**	**124**	💰

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

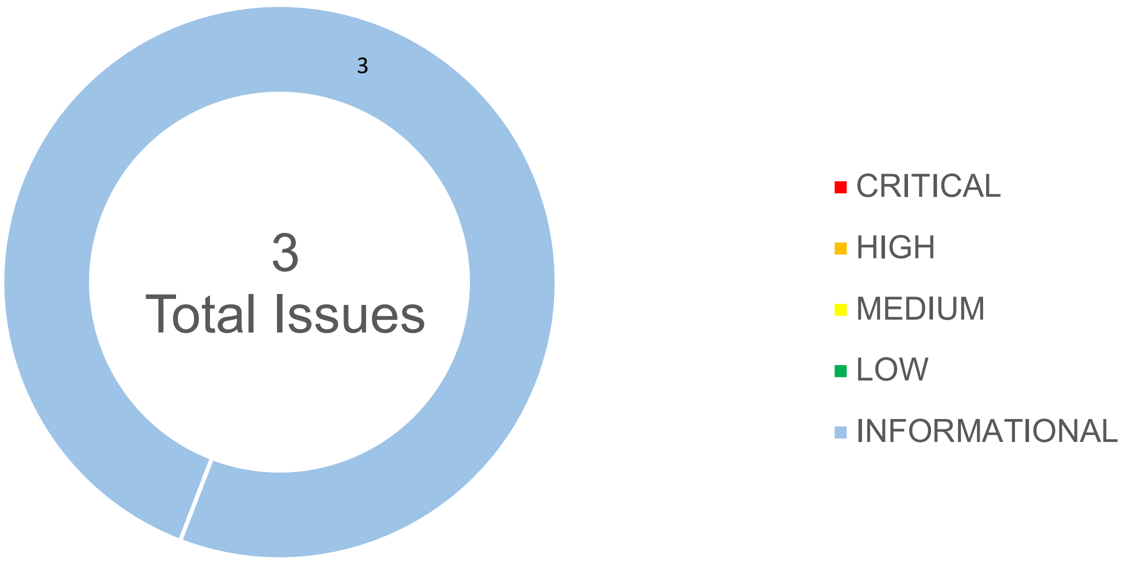
The Dispatch Team at IEX provided us with the files that needs to be tested. The scope of the audit is the dispatch core contract.

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

- The ERC-1155 Token standard is correctly implemented
- DispatchAdmin must be the Admin of the DispatchMessenger Contract
- Admin contract is approved to transfer anything in the Message contract, can burn messages in NFT contract and can burn many messages in NFT contract for one
- Message contract is not approved to transfer anything in the admin contract
- Non admin cannot send message NFTs or mint them
- Owner can add or remove other admins, can remove a TokenOrg and can set backendwallet
- Org Admin can add other admins or remove them and can mint message 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 And Different Pragma Versions Identified	INFORMATIONAL	FIXED
6.2.2	Consistent Naming Of TokenOrg	INFORMATIONAL	FIXED
6.2.3	Specific EVM Event Message	INFORMATIONAL	FIXED

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 And Different Pragma Versions Identified
Severity: INFORMATIONAL
Status: FIXED
Code: SWC-103
File(s) affected: ALL

Attack / Description	Code Snippet	Result/Recommendation

The current pragma solidity directive is "^0.8.7" and “^0.8.13”.	`pragma solidity ^0.8.7;` `pragma solidity ^0.8.13;`	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. i.e. Pragma solidity 0.8.13

6.2.2 Consistent Naming Of TokenOrg

Severity: INFORMATIONAL
Status: FIXED
Code: NA
File(s) affected: Admin.sol, Messenger.sol

Attack / Description	Code Snippet	Result/Recommendation

The usage of TokenOrg is not consistently used and can confuse the reader.	Line 49 (Admin.sol)
	`NFT does not belong to tokenOrg or it does not exist`
	
	Line 56 (Admin.sol)
	`A token org already exists at that address`
	
	Line 26 (Messenger.sol)
	`// mint N nfts to the _to address which will be the tokenOrgWallet`
		It is recommended to consistently use TokenOrg instead of tokenOrg, token org or tokenOrgWallet in events and nat spec.

6.2.3 Specific EVM Event Message

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

Attack / Description	Code Snippet	Result/Recommendation
The usage of specific event messages for each EVM is required or use generic.	Line 199 (Admin.sol) `require(sent, "Failed to send Matic");`	It is recommended to use “funds” instead of Matic, as the codebase is going to be used on different EVMs.

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	✅

ID	Title	Relationships	Test Result
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	✓
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✓

ID	Title	Relationships	Test Result
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	✅
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅

ID	Title	Relationships	Test Result
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	✅
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✅
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✅
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅

6.4 Verify Claims

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

6.4.2 DispatchAdmin must be the Admin of the DispatchMessenger Contract
Status: tested and verified ✅

6.4.3 Admin contract is approved to transfer anything in the Message contract, can burn messages in NFT contract and can burn many messages in NFT contract for one
Status: tested and verified ✅

6.4.4 Message contract is not approved to transfer anything in the admin contract
Status: tested and verified ✅

6.4.5 Non admin cannot send message NFTs or mint them
Status: tested and verified ✅

6.4.6 Owner can add or remove other admins, can remove a TokenOrg and can set backendwallet
Status: tested and verified ✅

6.4.7 Org Admin can add other admins or remove them and can mint message NFTs
Status: tested and verified ✅

6.4.8 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, no low and only 3 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

Ethereum
Dispatch Admin : https://etherscan.io/address/0xF6fd4D88E97da2a0a574b3adB9d95FA662DCA0Bc#code
Dispatch Messaging: https://etherscan.io/address/0xbC76cd93dBE8aCFd5Dd1583629eA210eA2df763e#code

Polygon
Dispatch Admin: https://polygonscan.com/address/0x6C903f20e2705aaaaC3F1B8c1cAa26E2c2131bd0#contracts
Dispatch Messaging: https://polygonscan.com/address/0x4E11Cb274981ab1D8Fce2A32d22975690A2F6deC#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