



Nearpad

Core

SMART CONTRACT AUDIT

28.09.2021

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
4.2 Used Code from other Frameworks/Smart Contracts	8
4.3 Tested Contract Files.....	9
4.4 Metrics / CallGraph	10
4.5 Metrics / Source Lines & Risk	11
4.6 Metrics / Capabilities.....	12
5. Scope of Work.....	15
5.1 Manual and Automated Vulnerability Test	16
5.1.1 Unverified smart contracts on-chain	16
5.1.2 Missing natspec documentation or outdated comments	17
5.2. SWC Attacks	18
5.3. Verify Claims	22
6. Executive Summary	23
7. Deployed Smart Contract.....	23



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 NEARPAD Project. 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 (26.08.2021)	Layout
0.2 (31.08.2021)	Test Deployment
0.5 (27.08.2021)	Automated Security Testing Manual Security Testing
0.6 (28.08.2021)	Testing SWC Checks
0.7 (01.09.2021)	Verify Claims
0.9 (02.09.2021)	Summary and Recommendation
1.0 (03.09.2021)	Final document
1.1 (14.09.2021)	Adding deployed contract address

2. About the Project and Company

Company address:

NEARPAD Project
KYC verified

Website: <https://nearpad.io>

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

Telegram: <https://t.me/nearpad>

Medium: <https://nearpad.medium.com>

Youtube: <https://www.youtube.com/channel/UCKXGYVket6ePTW0JUCayfmg>

GitHub: <https://github.com/nearpad-io>

2.1 Project Overview

NearPad is a protocol and a community - a protocol that enables users to connect to value and a community that adds value to projects. NearPad is uniquely based on making the protocol community-owned by vesting control of its treasury and public funds in the hands of its users. Holders are empowered to vote on projects and teams to incubate, developers can directly pitch their projects to the community.

A launchpad, DEX Aggregator, and Yield Aggregator all in one. NearPad is changing the way communities and developers access open finance tools for crowdfunding, asset management, and yield optimization. The platform will also be the first DAO-led protocol on Aurora by giving its community complete control over how treasury and public funds are utilised for the ecosystem.

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.



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

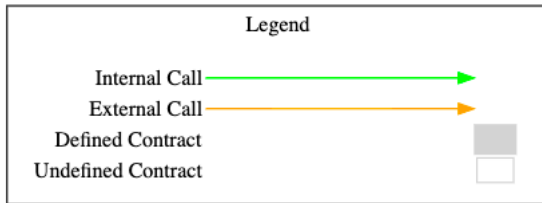
Dependency / Import Path	Source
@openzeppelin/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol
@openzeppelin/proxy/transparent/TransparentUpgradeableProxy.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/proxy/transparent/TransparentUpgradeableProxy.sol
@openzeppelin/proxy/utils/Initializable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/proxy/utils/Initializable.sol
@openzeppelin/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol
@openzeppelin/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol
@openzeppelin/token/ERC20/extensions/ERC20Pausable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC20Pausable.sol
@openzeppelin/token/ERC20/utils/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol
@openzeppelin/utils/Address.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Address.sol
@openzeppelin/utils/math/SafeMath.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/math/SafeMath.sol

4.3 Tested Contract Files

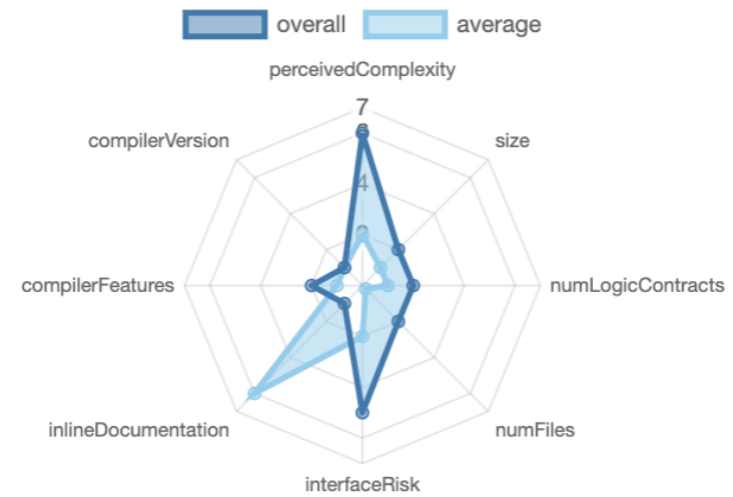
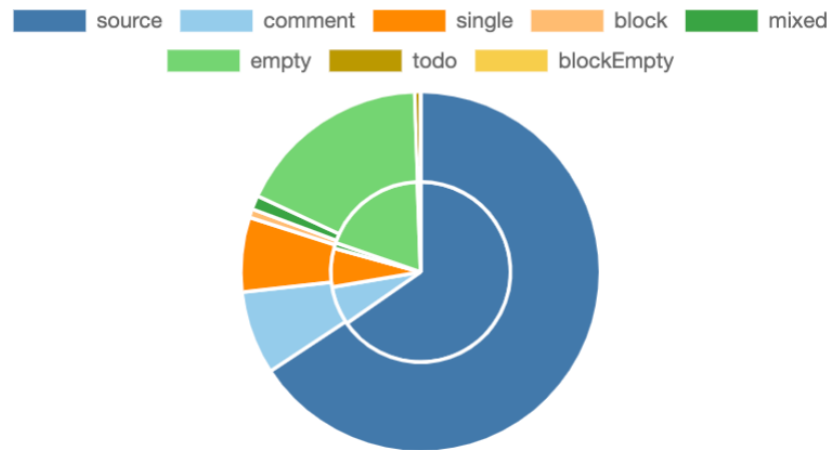
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability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

File	Fingerprint (MD5)
./common/PoolDetails.sol	28604e3be356beb1e957dc264d50b0b5
./helpers/HelperImpl.sol	83c24803daee62af9cc38918ce3ed538
./proxy/OwnableProxy.sol	96f74e8358e1023b38a5d89fb416dfa9
./proxy/PadProxy.sol	cf7bbcdbaa66b4fa5a38fb6f23cedc46
./staking/Staking.sol	ee2ab690a404e2f11eb26950fb1fd967
./token/NearPADToken.sol	f2cf10fdc7ead5b430de8045e0e84f0e
./token/VestingNearPADToken.sol	3e07afb0fd33b3aac7cdbdd8deca837b
./token/VestingTeamNearPADToken.sol	a2c0e6675ce4c19d09ab61ef6151a523
./FixedSwapRatePool.sol	8ae278ca7cc3258522c486f9e126e01b
./PoolFactory.sol	582ce851a8dace316555bce751d9ea9a
./PublicFixedSwapRatePool.sol	8119b4d51d4fd29c113e6baa9b2ce193









4.4 Metrics / CallGraph



4.5 Metrics / Source Lines & Risk





4.6 Metrics / Capabilities


Solidity Versions observed		 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
<code>^0.8.0</code>		<code>ABIEncoderV2</code>	<code>yes</code>	**** (0 asm blocks)	
 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				
74	2				
External	Internal	Private	Pure	View	
27	49	0	1	39	

StateVariables

Total	 Public
92	88

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/helpers/HelperImpl.sol	1	_____	75	51	41	2	115	
	contracts/FixedSwapRatePool.sol	1	_____	341	324	250	21	166	_____
	contracts/PoolFactory.sol	1	_____	135	120	67	27	80	_____
	contracts/staking/Staking.sol	1	_____	131	131	99	7	58	_____
	contracts/PublicFixedSwapRatePool.sol	1	_____	273	237	176	16	106	_____
	contracts/common/PoolDetails.sol	_____	_____	39	39	23	3	_____	_____
	contracts/proxy/OwnableProxy.sol	1	_____	15	15	10	1	9	
	contracts/proxy/PadProxy.sol	1	_____	15	15	9	2	7	
	contracts/token/VestingNearPADToken.sol	1	_____	167	155	112	18	72	_____
	contracts/token/VestingTeamNearPADToken.sol	1	_____	160	148	107	17	72	_____
	contracts/token/NearPADToken.sol	1	_____	26	26	17	1	14	_____

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	Totals	10		1377	1261	911	115	699	

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, ...)

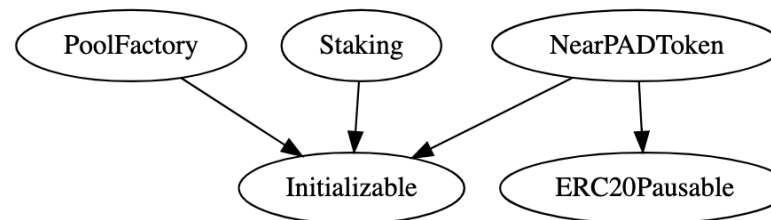
5. Scope of Work

The Nearpad Team provided us with the files that needs to be tested. The scope of the audit are the core contracts.

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

- Vesting of tokens is working as expected (Vesting)
- Launchpad Pools are working as expected (Pools / Staking)
- The Token is compliant with the ERC20 standard (Token)
- Owner cannot claim tokens before the buyer, out of the pool (Pools / Staking)
- Owner cannot pause the contract (Both)
- The smart contract is coded according to the newest standards and in a secure way. (Both)

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.



5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting's experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, Chainsulting's experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES

During the audit, Chainsulting's experts found **no Medium issues** in the code of the smart contract

LOW ISSUES

5.1.1 Unverified smart contracts on-chain

Severity: LOW

Status: **FIXED**

File(s) affected: NearPADToken.sol

Attack / Description	Code Snippet	Result/Recommendation
Unverified smart contracts can lead to FUD and can be seen as suspicious. User or researchers can't see if your smart contract has backdoors.	https://etherscan.io/address/0xea7cc765ebc94c4805e3bff28d7e4ae48d06468a#code	We recommend your project to verify the smart contract from your token.

INFORMATIONAL ISSUES

5.1.2 Missing natspec documentation or outdated comments

Severity: INFORMATIONAL

Status: **FIXED**

File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables and more. This special form is named the Ethereum Natural Language Specification Format (NatSpec).	NA	<p>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.</p> <p>There are already comments inside the codebase, but it can be increased.</p> <p>There are still unused comments or TODOs left, remove them before deployment to avoid confusion.</p>

5.2. 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	✓

5.3. Verify Claims


5.3.1 Vesting of tokens is working as expected (Vesting)

Status: tested and verified 


5.3.2 Launchpad Pools are working as expected (Pools / Staking)

Status: tested and verified 

5.3.3 The Token is compliant with the ERC20 standard (Token)

Status: tested and verified 

5.3.4 Owner cannot claim tokens before the buyer, out of the pool (Pools)


Status: tested and verified 

5.3.5 Owner cannot pause the contract (Both)

Status: tested and verified 

There is no function to pause the contracts

5.3.6 The smart contract is coded according to the newest standards and in a secure way.

Status: tested and verified 

6. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs took place on the September 03, 2021.

The main goal of the audit was to verify the claims regarding the security of the smart contract and the functions. During the audit, no critical issues were found after the manual and automated security testing and the claims been successfully verified.

7. Deployed Smart Contract

PAD = <https://etherscan.io/address/0xea7Cc765eBC94C4805e3BFf28D7E4aE48D06468A#code> VERIFIED

vPAD = <https://etherscan.io/address/0x5e87214ea27aac386832d051318f61950900583d#code> PENDING

vTPAD = <https://etherscan.io/address/0x7d350cf942f9892b1b9024c70d7af26b3a551207#code> PENDING

FACTORY = <https://etherscan.io/address/0x076fea47cbb2837bfba085b09bafac3a54bf8a72#code> VERIFIED

STAKING = <https://etherscan.io/address/0x1637B1Ccedb9c3f0d1c9C22A65C8A474b532a50F#code> VERIFIED

HELPER = <https://etherscan.io/address/0x20d5c04e8d644abc6638a47bae09fcf017853264#code> VERIFIED

PAD_PUBLIC_PAIR_01 = <https://etherscan.io/address/0xd526d1d84835cb6f06251b67920A4E812D7c4d1a#code> VERIFIED

