



NAFTA

NFT Flash Loans & Long-Term Renting without collateral

SMART CONTRACT AUDIT

06.07.2022

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.....	17
6.1 Findings Overview	18
6.2 Manual and Automated Vulnerability Test.....	19
6.2.1 A User Can Inject A Malicious Contract	19
6.2.2 Precision Loss Can Lead To Bypass Fees	20
6.2.3 Anyone Can Mint Infinite Amount Of Nafta NFTs	21
6.2.4 Missing Zero Address Checks	23
6.2.5 Missing Value Verification.....	23

6.2.6 Floating Pragma Version Identified	24
6.2.7 Renounce Ownership	25
6.3 SWC Attacks	26
6.4. Verify Claims	30
7. Executive Summary.....	31
8. Deployed Smart Contract	31
9. About the Auditor	32



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 NAFTA. 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 (10.06.2022)	Layout
0.4 (13.06.2022)	Automated Security Testing Manual Security Testing
0.5 (16.06.2022)	Verify Claims and Test Deployment
0.6 (20.06.2022)	Testing SWC Checks
0.9 (25.06.2022)	Summary and Recommendation
1.0 (26.06.2022)	Final document
1.1 (06.07.2022)	Added deployed contract

2. About the Project and Company

Company address:

Oiler DeFi Ltd.
c/o NAFTA
Jayla Place, 2nd Floor, Road Town
Tortola, British Virgin Islands, VG1110



Website: <https://www.nafta.market>

Documentation: <https://docs.oiler.network/oiler-network/products/nafta>

GitHub: <https://github.com/Nafta-Market>

2.1 Project Overview

Oiler is a protocol for blockchain native derivatives.

Before 2019/2020, it was practically impossible to deliver blockchain native derivatives. Without stablecoins and AMMs (automatic market makers), it was not possible to provide a reliable pricing solution. What has changed? Stablecoins introduced a non-volatile on-chain base for pricing (a USD peg) and AMMs introduced a pricing discovery mechanism; on-chain and with high volumes. It has also been proven that if the markets are efficient then the AMMs are efficient too and arbitrageurs will set the price right.

In order to settle derivatives on-chain nowadays, we need to ensure that the payout can be calculated entirely on-chain. At Oiler, they not only assume that they will not take off-chain data but also that there is no oracle hidden behind the layers of on-chain data sources that the smart contracts use. It means that the prices of the underlying instruments should not be derived from a protocol that uses off-chain oracles. Moreover, it is desirable to avoid any on-chain oracles like Uniswap since they can always be manipulated within a flash loan based attack. The last requirement is much stronger but still holds for the initial set of Oiler products.

Oiler network consist at the moment out of 3 products:

FOSSIL

Fossil is a set of smart contracts and infrastructure running on both Starknet and Ethereum L1. It's purpose is to enable starknet smart contracts to trustlessly access any data that ever appeared on ethereum such as: state, transactions, logs, block parameters. The purpose of Fossil API is to facilitate interacting with Fossil contracts. This API hides all the complexity from the end user behind a REST API and sends webhooks once the results are ready. The Fossil API is also able to batch different calls to Fossils together, to optimize our calls to L1.

NAFTA

Nafta is a protocol that facilitates NFT Flash Loans & Long Term Renting without any collateral. This is achieved by wrapping an original NFT when it is pooled with Nafta ecosystem. Hence rented assets are only used for the utilities they provide.

PITCH LAKE

DeFi vaults which allow you to trade Ethereum mainnet basefee. Pitch Lake uses StarkNet STARKSs to calculate basefee TWAP in a given month to be used for cash settlement. Access to verified Ethereum mainnet block header is achieved with the help of Oiler Fossil.

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

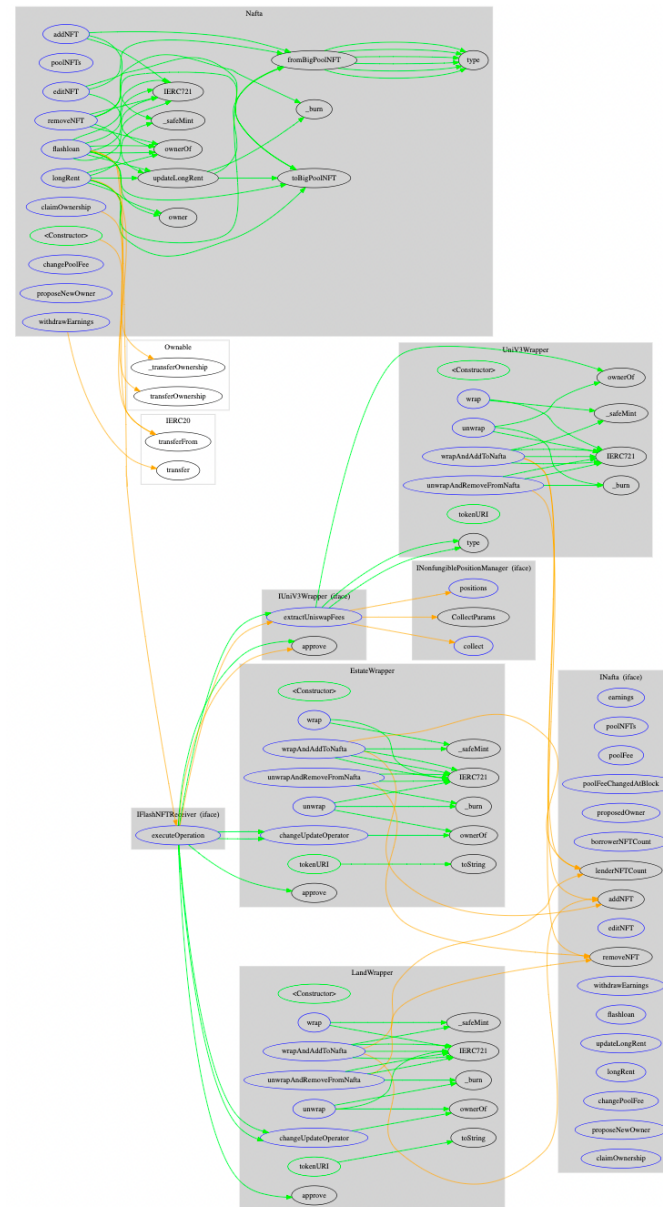
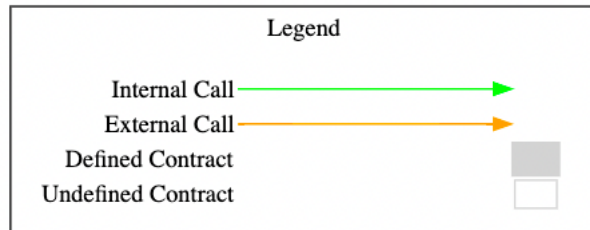
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability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

File	Fingerprint (MD5)
./contracts/interfaces/IFlashNFTReceiver.sol	48bab188a91762c5a6d450e195eceedf
./contracts/interfaces/INafta.sol	52b5c89f09ce7b16a757e7782f0b533f
./contracts/Nafta.sol	1a8185133ab9960b78a8f89cd0b3b950
./contracts/example/UniV3FlashLoan.sol	b61ed4beb95c7ab4eba73e2808aa8910
./contracts/example/IUniV3Wrapper.sol	8037b3364f53f6b1c8b59e3d9fe188f5
./contracts/example/UniV3Wrapper.sol	4b78d4d583d97320f899ad2daee65174
./contracts/example/INonfungiblePositionManager.sol	9b97471ed3fbf75ac89f4af53a811704
./contracts/example/EstateWrapper.sol	d1ab7d06fda858400014b6fe26e97fbc
./contracts/example/LandWrapper.sol	fe2e87b7e9fde0ce4983287e2bdd296b
./contracts/example/ILANDRegistry.sol	11f434df41a32387b7ba5dd3c5cc4f7a

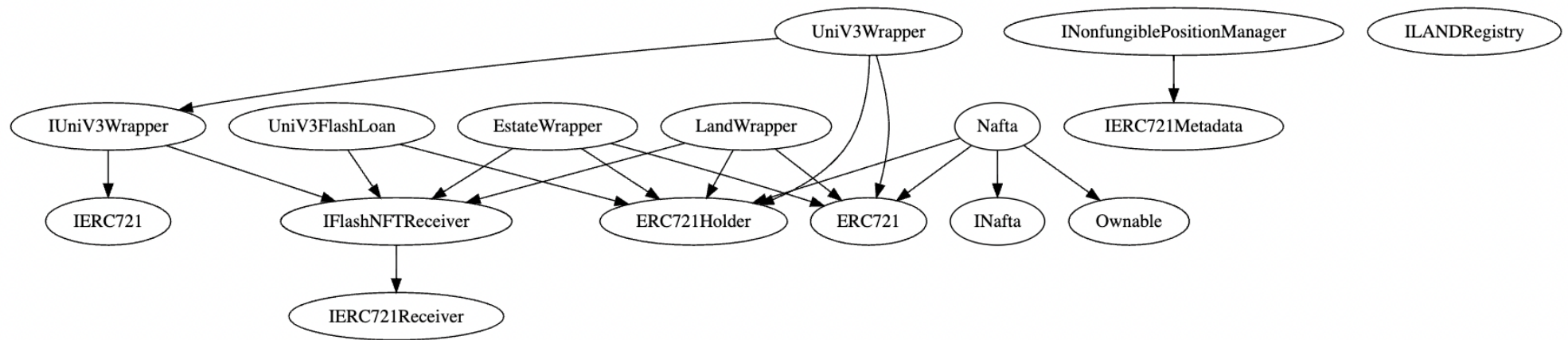
5.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0-rc.1/contracts/access/Ownable.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0-rc.1/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0-rc.1/contracts/token/ERC20/utils/SafeERC20.sol
@openzeppelin/contracts/token/ERC721/ERC721.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0-rc.1/contracts/token/ERC721/ERC721.sol
@openzeppelin/contracts/token/ERC721/IERC721.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0-rc.1/contracts/token/ERC721/IERC721.sol
@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0-rc.1/contracts/token/ERC721/IERC721Receiver.sol
@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0-rc.1/contracts/token/ERC721/extensions/IERC721Metadata.sol
@openzeppelin/contracts/token/ERC721/utils/ERC721Holder.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0-rc.1/contracts

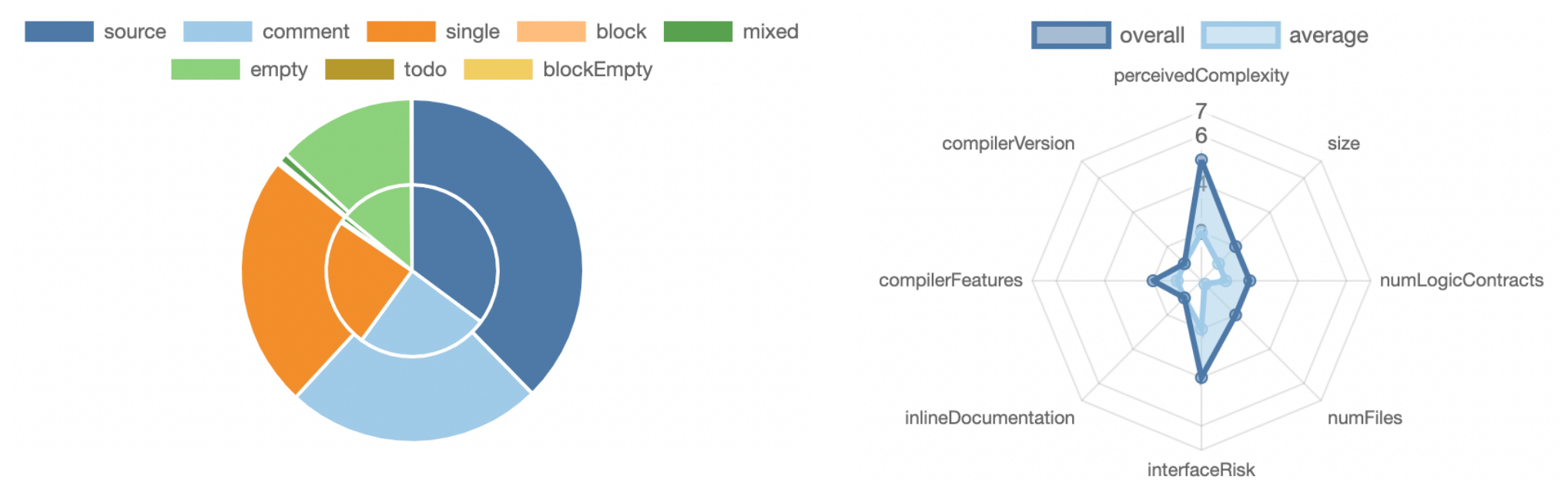
5.3 CallGraph













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	
<div><div>^0.8.9</div><div>>=0.7.5</div><div>0.8.13</div></div>		<div>yes</div>			
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECTRecover	 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				
58	1				
External	Internal	Private	Pure	View	
52	50	0	6	10	













StateVariables



Total	 Public
15	10

5.7 Source Unites in Scope

Source: <https://github.com/OilerNetwork/nafta>

Last commit: 773089f4768e726709ab2e8011718e2db9e1ab4c

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSL OC	Comment Lines	Complex. Score	Capabilities
	contracts/interfaces/IFlashNFTReceiver.sol	_____	1	8	7	4	1	5	_____
	contracts/interfaces/INafta.sol	_____	1	149	34	26	71	35	_____
	contracts/Nafta.sol	1	_____	404	380	162	135	161	
	contracts/example/UniV3FlashLoan.sol	1	_____	42	42	20	13	14	_____
	contracts/example/IUniV3Wrapper.sol	_____	1	13	12	5	4	7	_____
	contracts/example/UniV3Wrapper.sol	1	_____	158	142	73	40	77	_____
	contracts/example/INonfungiblePositionManager.sol	_____	1	60	40	10	26	10	
	contracts/example/EstateWrapper.sol	1	_____	173	157	84	47	94	_____
	contracts/example/LandWrapper.sol	1	_____	170	154	84	45	94	_____
	contracts/example/ILANDRegistry.sol	_____	1	14	10	3	7	5	_____

	Totals	5	5	119 1	978	471	389	502	
---	---------------	----------	----------	------------------	------------	------------	------------	------------	---

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

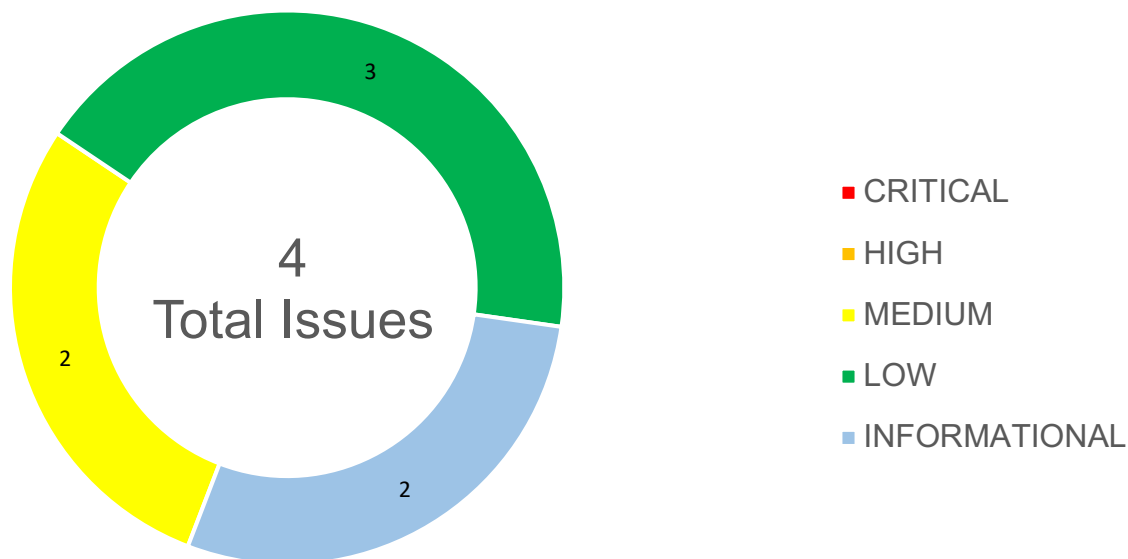
The Oiler Network Team provided us with the files that needs to be tested. The scope of the audit are the NAFTA contracts.

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

- The ERC-721 Token standard is correctly implemented
- Deployer cannot transfer or burn rented NFTs
- Long term and flashloan renting are working as intend
- Only the NFT Owner is able to withdraw earnings
- 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	A User Can Inject A Malicious Contract	MEDIUM	ACKNOWLEDGED
6.2.2	Precision Loss Can Lead To Bypass Fees	MEDIUM	ACKNOWLEDGED
6.2.3	Anyone Can Mint Infinite Amount Of Nafta NFTs	LOW	ACKNOWLEDGED
6.2.4	Missing Zero Address Checks	LOW	ACKNOWLEDGED
6.2.5	Missing Value Verification	LOW	ACKNOWLEDGED
6.2.6	Floating Pragma Version Identified	INFORMATIONAL	ACKNOWLEDGED
6.2.7	Renounce Ownership	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 **2 Medium issues** in the code of the smart contract.

6.2.1 A User Can Inject A Malicious Contract

Severity: MEDIUM

Status: ACKNOWLEDGED

Code: CWE-94

File(s) affected: Nafta.sol

Attack / Description	The addNFT function deposits the NFT of the user to the pool, by inserting the nftAddress and the nftId. A malicious user can deploy an own ERC721 contract with modified standard functions, such as transfer and transferFrom. Therefore, the user can call the addNFT function without sending any NFTs to the contract.
Code	Line 87 (Nafta.sol) <code>IERC721(nftAddress).safeTransferFrom(msg.sender, address(this), nftId);</code>
Result/Recommendation	Consider verifying if the user has sent the NFT by validating that the new owner is the contract, or consider verifying the NFT address to be included in the trusted assets.

6.2.2 Precision Loss Can Lead To Bypass Fees

Severity: MEDIUM

Status: ACKNOWLEDGED

Code: CWE-1339

File(s) affected: Nafta.sol

Attack / Description	In the longRent and flashLoan functions, the pool fee is calculated using the following formula $\text{poolFee} \times \text{longtermPayment} / 1e18$, the issue here is that if $\text{poolFee} \times \text{longTermPayment}$ is less than $1e18$ the poolPart will be equal to 0 due to loss precision.
Code	Line 232 (Nafta.sol) <pre>uint256 poolPart = (poolFee * lenderFees) / 1e18;</pre> Line 315 (Nafta.sol) <pre>uint256 poolPart = (poolFee * longtermPayment) / 1e18;</pre>
Result/Recommendation	Consider verifying that $\text{poolFee} \times \text{longTermPayment}$ is greater than $1e18$.

LOW ISSUES

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

6.2.3 Anyone Can Mint Infinite Amount Of Nafta NFTs

Severity: LOW

Status: ACKNOWLEDGED

Code: NA

File(s) affected: Nafta.sol

Attack / Description	The users can only mint Nafta's NFTs by locking their NFTs to the contract's pool. As there are no restrictions on the contract, the user can create an ERC721 contract and mint an infinite number of Nafta NFTs.
Code	<pre>Line 69-103 (Nafta.sol) function addNFT(address nftAddress, uint256 nftId, uint256 flashFee, uint256 pricePerBlock, uint256 maxLongtermBlocks) external { // Protection from weird and meaningless adding of NaftaNFTs require(nftAddress != address(this), "Can't add Nafta NFTs to Nafta"); // Verify that NFT isn't already in the pool require(_poolNFTs[nftAddress][nftId].lenderNFTId == 0, "NFT is already in the Pool"); // In Longterm rent, we have protection from cheaters who want to rent longterm for 1 block and pay less than a flash loan fee. // This check is to make sure if longrent with these parameters is even possible. // If pricePerBlock == 0 then Longterm rent is disabled. require((pricePerBlock == 0) (maxLongtermBlocks * pricePerBlock >= flashFee), "Max Longterm rent can't be cheaper than flashloan"); // Pull the NFT from the msg.sender using transferFrom IERC721(nftAddress).safeTransferFrom(msg.sender, address(this), nftId);</pre>

```

uint256 newNFTId = lenderNFTCount + 1;
lenderNFTCount = newNFTId;

// Store newly added NFT renting parameters after packing to struct
_poolNFTs[nftAddress][nftId] = fromBigPoolNFT(
    // (flashFee, pricePerBlock, maxLongtermBlocks, inLongtermTillBlock, borrowerNFTId,
    lenderNFTId)
    BigPoolNFT(flashFee, pricePerBlock, maxLongtermBlocks, 0, 0, newNFTId)
);

// Mint a new LenderNFT to msg.sender
_safeMint(msg.sender, newNFTId);

// Emit AddNFT event
emit AddNFT(nftAddress, nftId, flashFee, pricePerBlock, maxLongtermBlocks, newNFTId,
msg.sender);
}

```

Result/Recommendation

Its recommended to add a mapping of trusted NFT contracts, which are battle tested and 100% working with the renting protocol. Unforeseen contracts can contain malicious functions or not 100% working and cause loses.

6.2.4 Missing Zero Address Checks

Severity: LOW

Status: ACKNOWLEDGED

Code: NA

File(s) affected: Nafta.sol

Attack / Description	Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, some of the contract's functionality may become inaccessible. The owner, WETH9 arguments should be verified to be different from the address(0).
Code	Line 43 - 46 (Nafta.sol) <pre>constructor(address owner_, IERC20 WETH9_) ERC721("NaftaNFT", "NAFTA") { WETH9 = WETH9_; Ownable.transferOwnership(owner_); }</pre>
Result/Recommendation	It is recommended to verify that the addresses provided in the arguments are different than the address(0).

6.2.5 Missing Value Verification

Severity: LOW

Status: ACKNOWLEDGED

Code: CWE-345

File(s) affected: Nafta.sol

Attack / Description	Certain functions lack a value safety check, the values of the arguments should be verified to allow only the ones that comply with the contract's logic. In the editNFT function, the contract must ensure that flashFee is less than 4722.36648ETH.
-----------------------------	---

Code	Line 129 (Nafta.sol) bigPoolNFT.flashFee = flashFee;
Result/Recommendation	We recommend that you verify the values provided in the arguments. The issue can be addressed by utilizing a require statement.

INFORMATIONAL ISSUES

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

6.2.6 Floating Pragma Version Identified

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

Code: SWC-103

File(s) affected: ALL

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 ^0.8.9;</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 0.8.9
--	----------------------------

6.2.7 Renounce Ownership

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

Code: NA

File(s) affected: Nafta.sol

Attack / Description	Generally, 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 (Nafta.sol) <code>contract Nafta is INafta, ERC721, ERC721Holder, Ownable {</code>
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.

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	X
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-721 Token standard is correctly implemented

Status: tested and verified 


6.4.2 Deployer cannot transfer or burn rented NFTs

Status: tested and verified 


6.4.3 Long term and flashloan renting are working as intend

Status: tested and verified 

6.4.4 Only the NFT Owner is able to withdraw earnings

Status: tested and verified 

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 and functions of the smart contract. During the audit, no critical, no high, 2 medium, 3 low and 2 informational issues have been found, after the manual and automated security testing.

We advise the Oiler Network team to implement the recommendations contained in all 7 of our findings, to further enhance the code's security. It is of utmost priority to start by addressing the most severe issues discovered by the auditors, then followed by the remaining issues. Afterwards, we will conducting a re-check following the implementation of the remediation plan contained in this report.

8. Deployed Smart Contract

VERIFIED

Mainnet:

Nafta: <https://etherscan.io/address/0x5D097eBfc47A2fC41B4A3cb16fDD6bc93435Fbdd#code>

UniV3Wrapper: <https://etherscan.io/address/0x2Ef97d5f5b55561f391EbFb3C8c277fFd7e34635#code>

EstateWrapper: <https://etherscan.io/address/0xe49a52B093E5c2a99D2DB7F47Ba649E3884A6790#code>

LandWrapper: <https://etherscan.io/address/0xC9582031c94BC97D1CeF466A092324B137393172#code>

Polygon:

Nafta: <https://polygonscan.com/address/0x5D097eBfc47A2fC41B4A3cb16fDD6bc93435Fbdd#code>

UniV3Wrapper: <https://polygonscan.com/address/0x2Ef97d5f5b55561f391EbFb3C8c277fFd7e34635#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>

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.