GAIA WORLD

LAND NFT

SMART CONTRACT AUDIT

31.03.2022

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 Omnisoft LTD (GAIA Everworld). 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 (26.01.2022)	Layout
0.2 (27.01.2022)	Test Deployment
0.5 (28.01.2022)	Automated Security Testing Manual Security Testing
0.6 (30.01.2022)	Testing SWC Checks
0.7 (31.01.2022)	Verify Claims
0.9 (01.02.2022)	Summary and Recommendation
1.0 (02.02.2022)	Final document
1.1 (25.03.2022)	Re-check
1.2 (31.03.2022)	Re-check
1.3 (31.03.2022)	Added deployed contract addresses

2. About the Project and Company

Omnisoft LTD
OMC Chambers
Wickhams Cay 1
Road Town, Tortola
British Virgin Islands

Website: https://gaiaworld.com

Twitter: https://twitter.com/GaiaEverWorld

Medium: https://medium.com/@gaia-world

Telegram: https://t.me/GaiaEverworld

Discord: https://discord.gg/EGT7c4RVfs

Email: contact@gaiaworld.com

2.1 Project Overview

Gaia Everworld blends classic fantasy narratives with state of the art blockchain and NFT technology. In the multi-realm gaming environment, players will be able to use their Gaia Legionnaires to wage campaigns, defend lands, and other immersive activities. Like many other games, like Pokemon, or Clash of Clans, Gaia Everworld allows players to own their characters, and interact in a dynamic environment with other human players all over the world.

The gaming environment allows for players to choose a homeland, which will give their NFT-based Gaia special powers, as well as weaknesses. The game uses a play-to-earn model, so that players have a financial incentive to join and play.

In Gaia Everworld, they offer players the ability to exist in a multi-realm online environment and participate in both PVP Battles and Legion Mode. The game centers on Gaia — a mythical creature that can be bred and owned in the form of an NFT.
The underlying goal of the game is to have the strongest collection of Gaia. With these NFT creatures, players can battle other players in the game, and conquer the lands of Gaia Everworld.
Of course, Gaia can be bred and added to a collection of other Gaia — or sold to other players. The two tokens that make the platform work are $GAIA, which can be staked, and $GGP, which is needed to breed Gaia.

- Holders of $GAIA can stake coins to earn $GAIA.
- Players to earn $GAIA and $GGP (Gaia Growth Potion) by playing the game and participating in events and adventures.
- Players to trade or sell their Gaia, Gaia eggs, land and resources in the Gaia Everworld marketplace.
- Players to loan their Gaia to other players in a peer to peer contract. The owner then earns a percentage of the $GGP earned by the loanees game play.

With NFT based games, players are also the owners of the game, and can control the platform to a much higher level than ever before.

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/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/access/Ownable.sol
@/openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/introspection/IERC165.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/introspection/ERC165.sol
@openzeppelin/contracts/token/ERC721/IERC721Metadata.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/token/ERC721/IERC721Metadata.sol
@openzeppelin/contracts/token/ERC721/IERC721Enumerable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/token/ERC721/IERC721Enumerable.sol
@openzeppelin/contracts/token/ERC721/IERC721.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/token/ERC721/IERC721.sol
@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/token/ERC721/IERC721Receiver.sol
@openzeppelin/contracts/introspection/ERC165.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/introspection/ERC165.sol
@openzeppelin/contracts/math/SafeMath.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/math/Math.sol

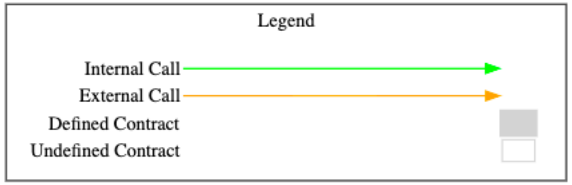
Dependency / Import Path	Source
@openzeppelin/contracts/utils/Address.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/utils/Address.sol
@openzeppelin/contracts/utils/EnumerableSet.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/utils/EnumerableSet.sol
@openzeppelin/contracts/utils/EnumerableMap.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/utils/EnumerableMap.sol
@openzeppelin/contracts/token/ERC721/ERC721.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/token/ERC721/ERC721.sol
@openzeppelin/contracts/utils/Context.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/utils/Context.sol
@openzeppelin/contracts/utils/Strings.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/utils/Strings.sol
@openzeppelin/contracts/cryptography/MerkleProof.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/solc-0.6/contracts/cryptography/MerkleProof.sol

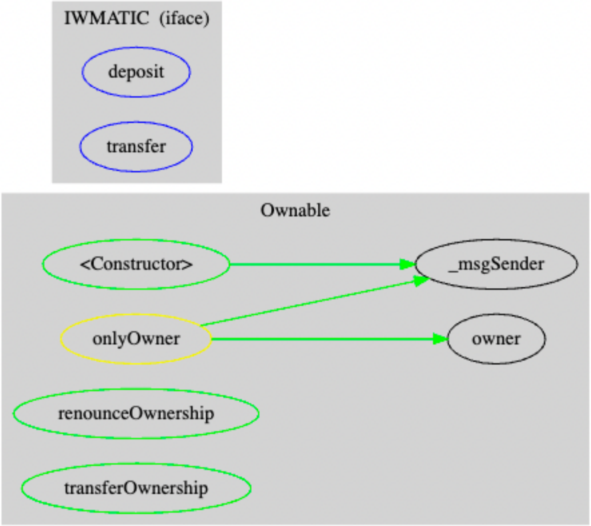
4.3 Tested Contract Files

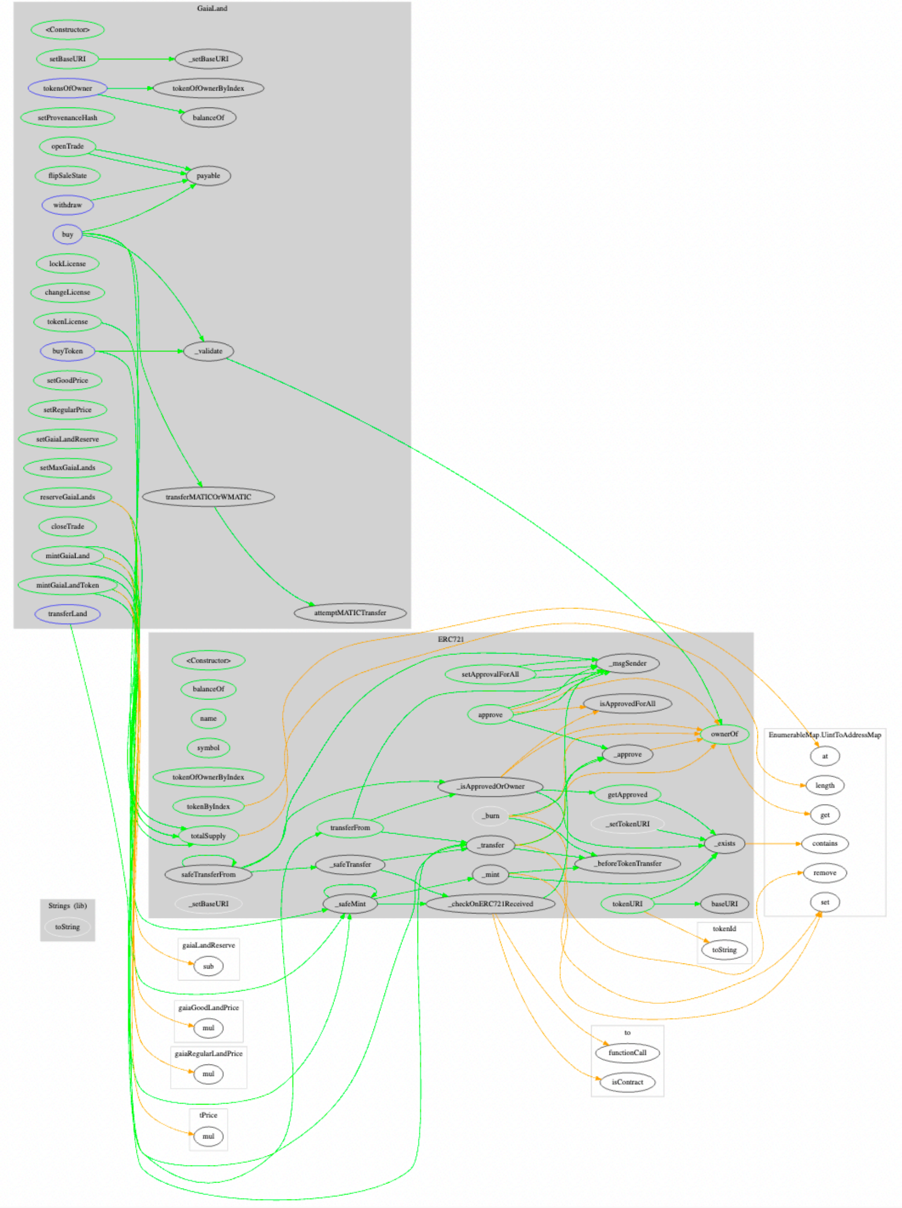
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability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

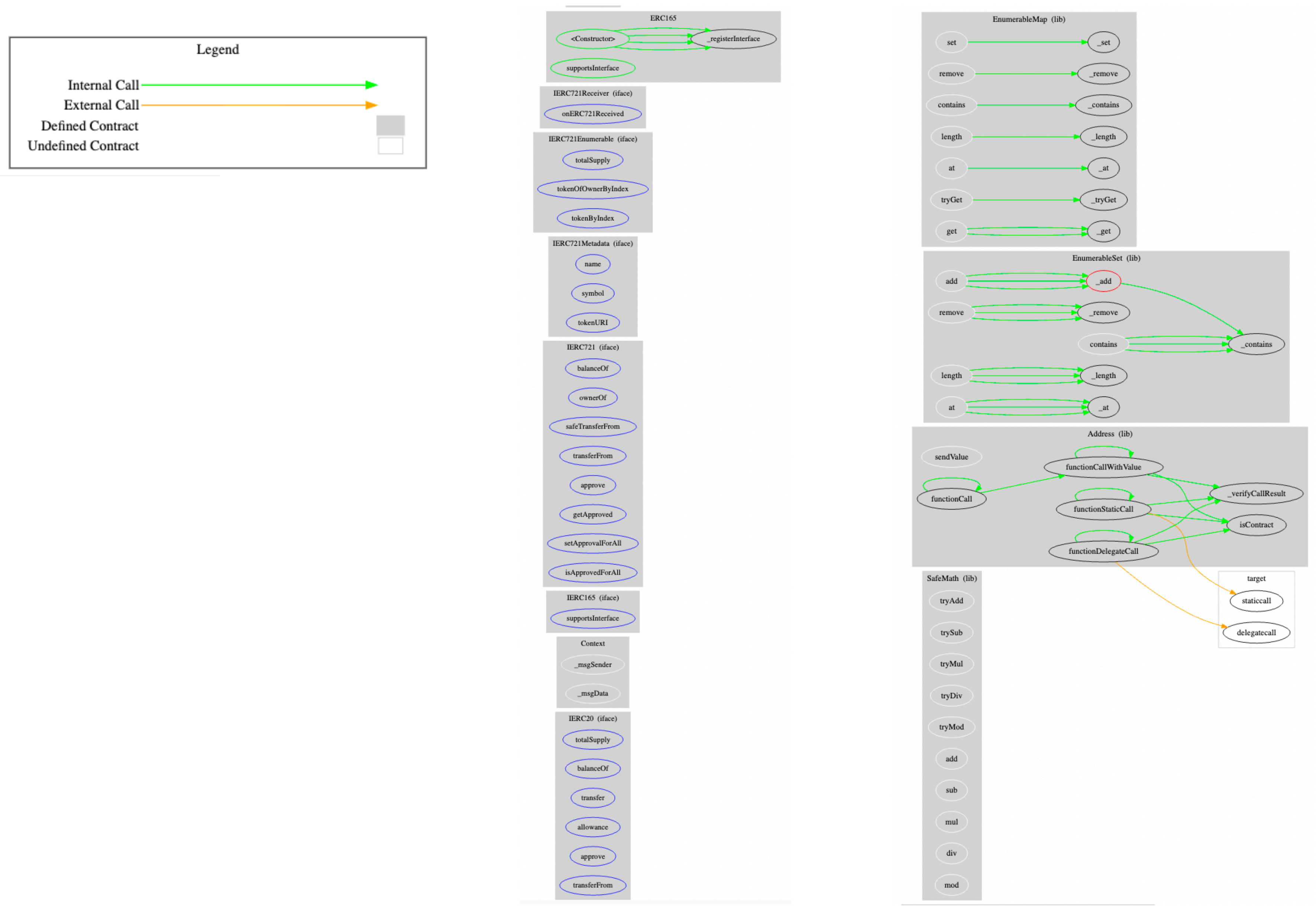
File	Fingerprint (MD5)
GAIALand.sol	2fe9985d48dcbbeca589fa668b193731
GAIALand.sol (25.03.2022)	a564a05370901a3f83c1b56b0090b883
GAIALand.sol (31.03.2022)	331d2a206c1323f75135ef5b438b0022

4.4 Metrics / CallGraph

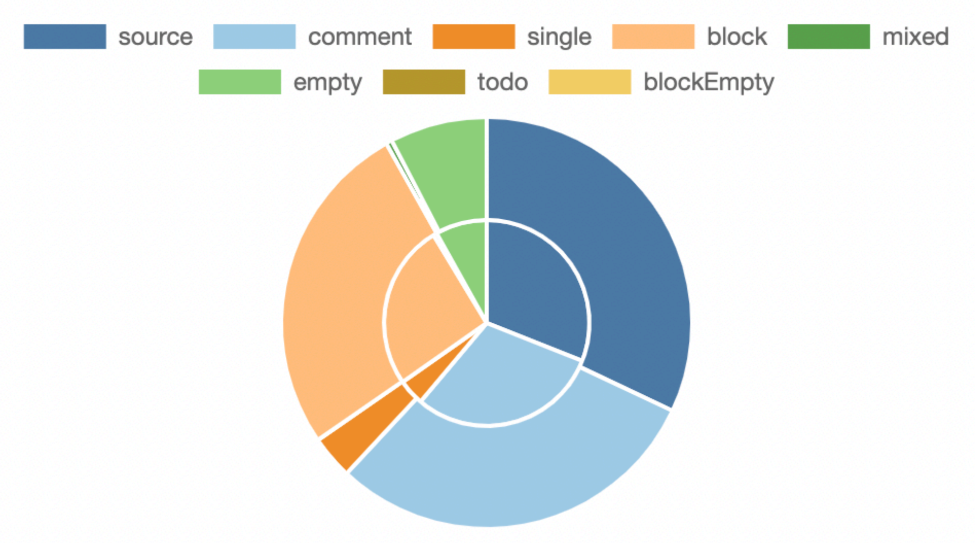


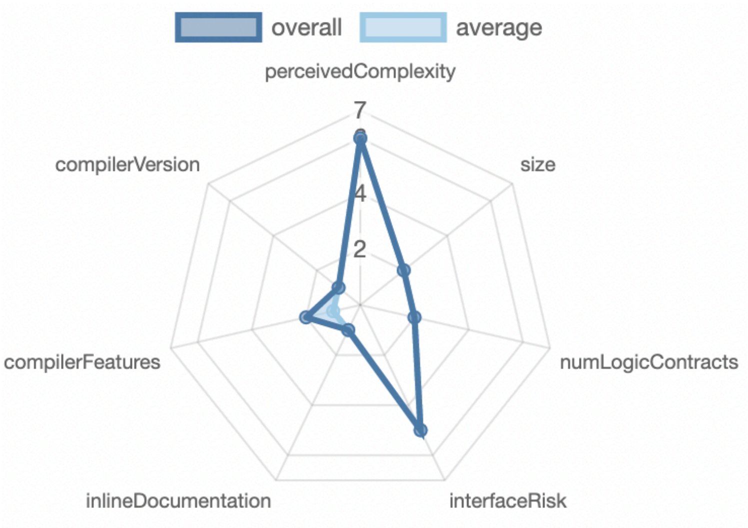





Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ERC165
<Constructor>
_registerInterface
supportsInterface
IERC721Receiver (iface)
onERC721Received
IERC721Enumerable (iface)
totalSupply
tokenOfOwnerByIndex
tokenByIndex
IERC721Metadata (iface)
name
symbol
tokenURI
IERC721 (iface)
balanceOf
ownerOf
safeTransferFrom
transferFrom
approve
getApproved
setApprovalForAll
isApprovedForAll
IERC165 (iface)
supportsInterface
Context
_msgSender
_msgData
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
EnumerableMap (lib)
set
_set
remove
_remove
contains
_contains
length
_length
at
_at
tryGet
_tryGet
get
_get
EnumerableSet (lib)
add
_add
remove
_remove
contains
_contains
length
_length
at
_at
Address (lib)
sendValue
functionCallWithValue
functionCall
_verifyCallResult
functionStaticCall
isContract
functionDelegateCall
target
staticcall
delegatecall
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod

4.5 Metrics / Source Lines & Risk





4.6 Metrics / Capabilities

Solidity Versions observed	🧪 **Experimental Features**	💰 **Can Receive Funds**	🖥 **Uses Assembly**	💣 **Has Destroyable Contracts**

`>=0.6.0 <0.8.0`
`>=0.6.2 <0.8.0`
`^0.7.0`
		`yes`	`yes`
			(2 asm blocks)	

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

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**
65	3

External	**Internal**	**Private**	**Pure**	**View**
30	133	17	15	61

StateVariables

Total	🌐**Public**
31	14

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚🔍🎨	GAIALand.sol	10	7	2564	2041	921	1110	625	🖥💰📤👥🔆
📝📚🔍🎨	**Totals**	**10**	**7**	**2564**	**2041**	**921**	**1110**	**625**	🖥💰📤👥🔆

Update 31st of March

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚🔍🎨	gaia_landv3.sol	11	7	2719	2159	1019	1116	690	🖥💰📤👥🧮🔆
📝📚🔍🎨	**Totals**	**11**	**7**	**2719**	**2159**	**1019**	**1116**	**690**	🖥💰📤👥🧮🔆

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, ...)

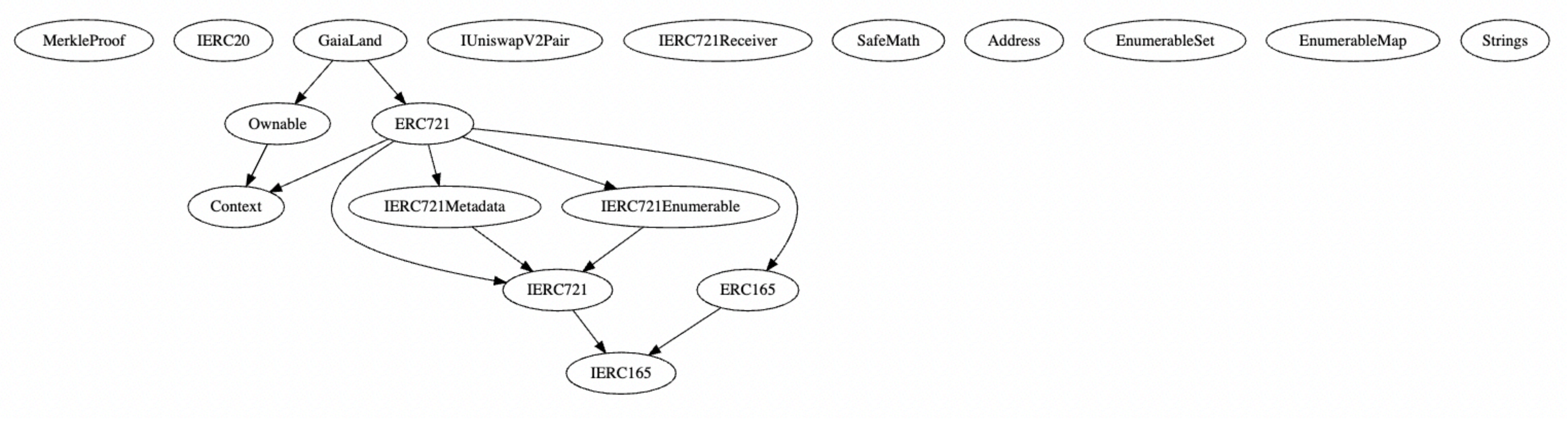
5. Scope of Work

The GAIA EverWorld Team provided us with the files that need to be tested. The scope of the audit is the GAIA NFT Land contract.

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

- The GAIA NFT Land is correctly implemented with the ERC721 Standard
- Owner cannot mint new land after minting was done
- Owner cannot burn land
- Owner is not able to pause the contract
- Minting of Land is random and can’t be front run
- Mathematical calculations inside the contract are correctly performed
- 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.



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

5.1.1 Overpowered Owner rights
Severity: MEDIUM
Status: FIXED
Code: NA
File(s) affected: GAIALand.sol

Attack / Description	Code Snippet	Result/Recommendation

Owners can perform privileged activities like withdraw funds from contract, reserve GAIALANDS to any address(can effectively burn by sending to zero address), pause contracts by making sale inactive, set the price, set maximum reserve and set maximum GAIALANDS (effectively being able to mint) . The auditor has not recognized any multi sig structure.	`function setMaxGaiaLands(uint256`
	`newMaxGaiaLands) public onlyOwner {`
	`    MAX_GAIALANDS = newMaxGaiaLands;`
	`  }`
	
	`function reserveGaiaLands(address _to, uint256`
	`_reserveAmount)`
	`    public`
	`    onlyOwner`
	`  {`
	`    uint256 supply = totalSupply();`
	`    require(`
	`      _reserveAmount > 0 && _reserveAmount <=`
		It is recommended to use multisig wallet for owner privileges.
		
		Owner is able to pause contract via ability to call flipSaleState() function to make isActive(true or false) at any time without restriction

	gaiaLandReserve,
	    'Not enough reserve left for team'
	  );
	  for (uint256 i = 0; i < _reserveAmount; i++)
	{
	    _safeMint(_to, supply + i);
	  }
	  gaiaLandReserve =
	gaiaLandReserve.sub(_reserveAmount);
	 }	

LOW ISSUES

5.1.2 Variable could be declared constant

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: GAIALand.sol

Attack / Description	Code Snippet	Result/Recommendation
It is recommendations to define MAX_GAIALANDS as constant variable.	uint256 public MAX_GAIALANDS = 124884;	It is recommended to define constant variables properly with the constant keyword to improve code readability.

5.1.3 Public functions should be declared as external

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: GAIALand.sol

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation several functions are declared as public where they could be external. For public functions Solidity immediately copies array arguments to memory, while external functions can read directly from calldata. Because memory allocation is expensive, the gas consumption of public functions is higher.	`function reserveGaiaLands(address _to, uint256 _reserveAmount)`
	`public`
	`onlyOwner`
	
	`function tokenLicense(uint256 _id) public view returns (string memory)`
	
	`function setGoodPrice(uint256 newPrice) public onlyOwner`
	
	`function setRegularPrice(uint256 newPrice) public onlyOwner`
	
	`function setMaxGaiaLands(uint256 newMaxGaiaLands) public onlyOwner`
	
	`function setGaiaLandReserve(uint256 newGaiaLandReserve) public onlyOwner`
	
	`function mintGaiaLand(`
	`uint256 numberOfTokens,`
		We recommend declaring functions as external if they are not used internally. This leads to lower gas consumption and better code readability.

	uint256 numberOfGoodLands, uint256 numberOfRegularLands) public payable function mintGaiaLandToken(uint256 numberOfTokens, uint256 numberOfGoodLands, uint256 numberOfRegularLands, uint256 numberOfRate) public function openTrade(uint256 _id, uint256 _price, uint256 duration, string memory unit, uint256 mintId) public function closeTrade(uint256 _id) public	

5.1.4 State visibility is not set

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: GAIALand.sol

Attack / Description	Code Snippet	Result/Recommendation

State variables without visibility set.	`bool licenseLocked = false;`
	`address ownerAddress;`
		State variable must be declared internal, private or public.

5.1.5 Redundant code

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: GAIALand.sol

Attack / Description	Code Snippet	Result/Recommendation

The current implementation has a redundant code in function buy() and buyToken()	`listedMap[_id] = false;` in functions
	`function buy(uint256 _id, uint256 _price)`
	`external payable`
	`function buyToken(uint256 _id, uint256 _price)`
	`external`
		It is highly recommended to remove `listedMap[_id] = false;` the redundant assignments for mappings as it unnecessarily increases code weight. Mappings have default values when looking up non-existing keys. For example mapping(address => bool) x a lookup of x[address] will return false if address does not exist.

5.1.6 Missing zero address checks

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: GAIALand.sol

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation several functions are not checking for zero addresses. Setting an address to the zero address can result in loosing funds by sending it to the zero address.	`function reserveGaiaLands(address _to, uint256 _reserveAmount)`
	`public`
	`onlyOwner`
	
	`function transferLand(uint256 _id, address to) external`
		It is highly recommended to check address e.g require(_address != address(0))

5.1.7 Missing Events

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: GAIALand.sol

Attack / Description	Code Snippet	Result/Recommendation

Several functions will benefit from having events, which are allowing a proper tracking, logging in some cases.	`function setProvenanceHash(string memory provenanceHash) public onlyOwner {`
	`GAIALANDS_PROVENANCE = provenanceHash;`
	`}`
	`function setGoodPrice(uint256 newPrice) public onlyOwner {`
	`gaiaGoodLandPrice = newPrice;`
		It is recommended to emit events for these critical functions to allow tracking, monitoring, logging and alerting.

```
 }

function setRegularPrice(uint256 newPrice)
public onlyOwner {
   gaiaRegularLandPrice = newPrice;
 }

function setGaiaLandReserve(uint256
newGaiaLandReserve) public onlyOwner {
   gaiaLandReserve = newGaiaLandReserve;
 }

function setMaxGaiaLands(uint256
newMaxGaiaLands) public onlyOwner {
   MAX_GAIALANDS = newMaxGaiaLands;
 }

 function withdraw() external onlyOwner {
   uint256 balance = address(this).balance;
   payable(ownerAddress).transfer(balance);
 }
```

5.1.8 Long number literals

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: GAIALand.sol

Attack / Description	Code Snippet	Result/Recommendation

Long number literals hardcoded in the contracts are prone to errors.	`uint256 public gaiaGoodLandPrice = 7500000000000000;`
	`uint256 public gaiaRegularLandPrice = 2500000000000000;`
		It is highly recommended long number literals should be checked and written in scientific notation
		e.g 1e18 vs 1000000000000000000

5.1.9 No return value checks

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: GAIALand.sol

Attack / Description	Code Snippet	Result/Recommendation

Contract may fail or work unexpectedly if return values are not checked.	`function transferMATICOrWMATIC(address payable to, uint256 value) private {`
	`if (!attemptMATICTransfer(to, value)) {`
	`IWMATIC(WMATICAddress).deposit{value: value}();`
	`IWMATIC(WMATICAddress).transfer(to, value);`
	`}`
	`}`
		It is highly recommended to check return values, especially low level calls or functions that return something. IWMATIC(WMATICAddress).transfer() function returns a bool that must be checked and can wrap in a require.
		`require(IWMATIC(WMATICAddress).transfer(to, value));`

5.1.10 Inefficient use of structs

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: GAIALand.sol

Attack / Description	Code Snippet	Result/Recommendation

Structs not properly used in terms of variable sizes, and ordering may result in higher gas costs.	`struct Auction {`
	`uint256 price;`
	`string unit;`
	`uint256 duration;`
	`uint256 startTime;`
	`uint256 endTime;`
	`bool status;`
	`uint256 id;`
	`uint256 mintId;`
	`address creator;`
	`address payable newOwner;`
	`address payable preOwner;`
	`}`
		It is recommended to take advantage of storage packing where values smaller than 256 bits are stored in the same storage slot.
		uint256 duration can be made uint32
		uint256 startTime can be made uint32
		uint256 endTime can be made uint32
		uint256 id can be made uint32
		uint mintId can be made uint32
		The above variables can all be packed in one storage slot leading to gas savings.

INFORMATIONAL ISSUES

5.1.11 Missing natspec documentation

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: GAIALand.sol

Attack / Description	Code Snippet	Result/Recommendation
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).	`//…`	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.

5.1.12 Floating compiler versions

Severity: INFORMATIONAL
Status: FIXED
Code: SWC-103
File(s) affected: GAIALand.sol

Attack / Description	Code Snippet	Result/Recommendation
The current pragma solidity directive is floating. 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.	`pragma solidity ^0.7.0;`	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. i.e. Pragma solidity 0.7.0 See SWC-103: https://swcregistry.io/docs/SWC-103

5.1.13 Using newest compiler version

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

Attack / Description	Code Snippet	Result/Recommendation
A higher compiler version has in most cases new features implemented or bugs fixed.	`pragma solidity ^0.7.0;`	It is recommended to use the stable version 0.8.4 where abicoderv2 added and overflow underflow checked automatically

5.2 SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	X
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	✅

5.3. Verify Claims

5.3.1 The GAIA NFT Land is correctly implemented with the ERC721 Standard
Status: tested and verified ✅

5.3.2 Owner cannot mint new land after minting was done
Status: tested and verified ✅

5.3.3 Owner cannot burn land
Status: tested and verified ✅

5.3.4 Owner is not able to pause the contract
Status: Owner is able to pause contract via ability to call flipSaleState() function to make isActive(true or false) at any time without restriction.

5.3.5 Minting of Land is random and can’t be front run
Status: tested and verified ✅

5.3.6 Mathematical calculations inside the contract are correctly performed
Status: tested and verified ✅

5.3.7 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 January 28, 2022.

Main goal of the audit was to verify the claims regarding the security of the smart contract. During the audit, 1 Medium, 9 Low and 3 Informational issues were found, after the manual and automated security testing and not all claims have been successfully verified. Please check all issues and get back to your auditor when issues have been fixed.

Update: https://polygonscan.com/address/0xF6e3c3184c9e58D2d3d520B7D7D79feE1B5E8732#code Re-check has been done at the 25th of March 2022.

Update: https://polygonscan.com/address/0xa09795ec053826ecea14ca48346327bb33e90a78#code Re-check has been done at the 31st of March 2022. Merkle tree has been added for whitelisting’s

7. Deployed Smart Contract

VERIFIED
https://polygonscan.com/address/0x1cc11c94ea60a01a5aa3c67815dc63581afb0802#code

8. About the Auditor

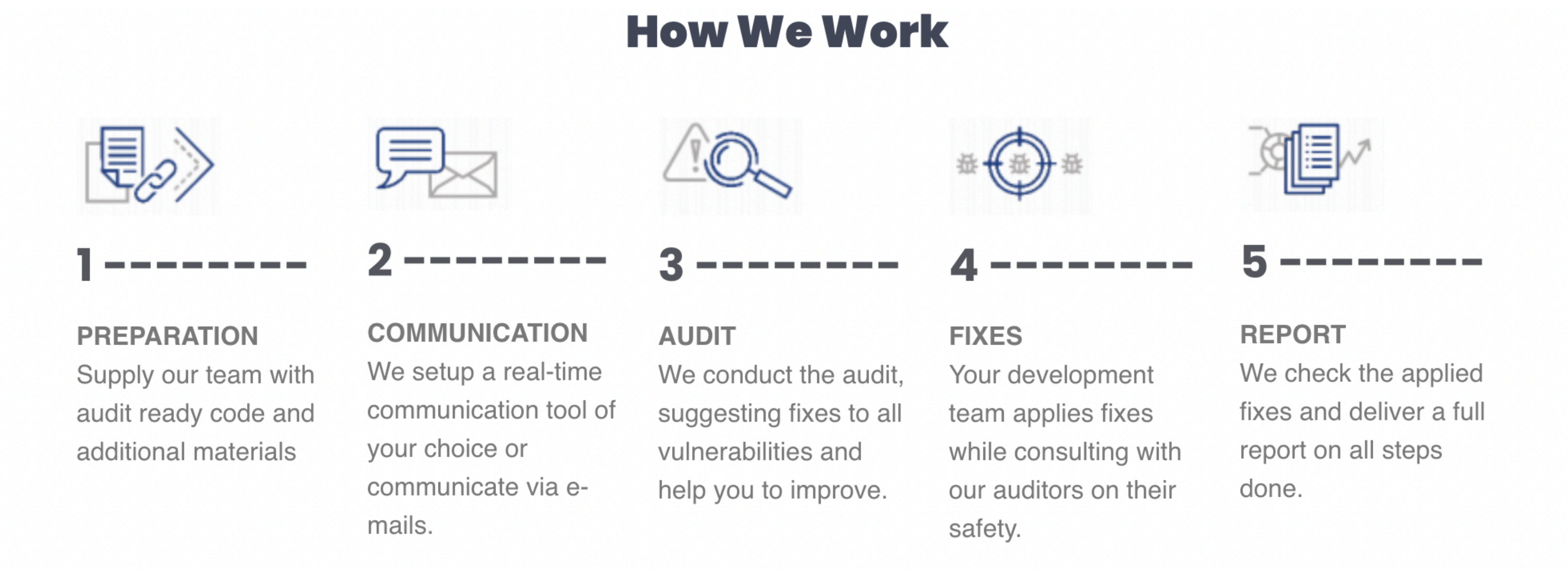
Chainsulting is a professional software development firm based in Germany that provides comprehensive distributed ledger technology (DLT) solutions. Some of their services include blockchain development, smart contract audits and consulting.

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.

The blockchain security provider brings the highest security standards to crypto and blockchain platforms, helping to foster growth and transparency within the fast-growing ecosystem.

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.