InterFi NETWORK

@INTERFINETWORK

SMART CONTRACT SECURITY AUDIT OF

CRYPTO WALKERS WEAPONS

SMART CONTRACT AUDIT | SOLIDITY DEVELOPMENT & TESTING | PROJECT EVALUATION

RELENTLESSLY SECURING THE PUBLIC BLOCKCHAIN

Audit Introduction

Auditing Firm	InterFi Network
Audit Architecture	InterFi Echelon Auditing Standard
Language	Solidity
Client Firm	Crypto Walkers Weapons
Website	
Telegram	
Twitter	
Discord	
Report Date	April 01, 2022

About Crypto Walkers Weapons

Audit Summary

InterFi team has performed a line-by-line manual analysis and automated review of smart contracts. Smart contracts were analyzed mainly for common contract vulnerabilities, exploits, and manipulation hacks. According to the audit:

- Crypto Walkers Weapons' solidity source code has **LOW RISK SEVERITY**
- Crypto Walkers Weapons' smart contract has an **ACTIVE OWNERSHIP**
- Crypto Walkers Weapons' centralization risk correlated to the active owner is **LOW**
- Important owner privileges – **ADMIN MINT, SET STATE, WITHDRAW ALL**

Be aware that smart contracts deployed on the blockchain aren't resistant to internal exploit, external vulnerability, or hack. For a detailed understanding of risk severity, source code vulnerability, exploitability, and audit disclaimer, kindly refer to the audit.

🚨 Contract address: **Not deployed**

⛓️ Blockchain: **Not chained**

✅ Verify the authenticity of this report on InterFi's GitHub: **https://github.com/interfinetwork**

Table Of Contents

Audit Information

Audit Scope 5

Echelon Audit Standard

Audit Methodology 6

Risk Classification 8

Centralization Risk 9

Smart Contract Risk Assessment

Static Analysis 10

Software Analysis 12

Manual Analysis 14

SWC Attacks 16

Risk Status & Radar Chart 18

Audit Summary

Auditor's Verdict 19

Legal Advisory

Important Disclaimer 20

About InterFi Network 21

Audit Scope

InterFi was consulted by Crypto Walkers Weapons to conduct the smart contract security audit of their solidity source codes. The audit scope of work is strictly limited to the mentioned solidity file(s) only:

- CryptoWalkersWeapons.sol

Solidity Source Code On GitHub

//Private Repository//

SHA-1 Hash

Solidity source code is audited at hash #9aaa15eed1802b458c92fdde13051a1385e41011

Audit Methodology

The scope of this report is to audit the smart contract source code of Crypto Walkers Weapons. InterFi has scanned contracts and reviewed codes for common vulnerabilities, exploits, hacks, and back-doors. Due to being out of scope, InterFi has not tested contracts on testnet to assess any functional flaws. Below is the list of commonly known smart contract vulnerabilities:

Category	
Smart Contract Vulnerabilities	❖ Re-entrancy ❖ Unhandled Exceptions ❖ Transaction Order Dependency ❖ Integer Overflow ❖ Unrestricted Action ❖ Incorrect Inheritance Order ❖ Typographical Errors ❖ Requirement Violation ❖ Gas Limit and Loops
Source Code Review	❖ Deployment Consistency ❖ Repository Consistency ❖ Data Consistency ❖ Token Supply Manipulation ❖ Access Control and Authorization ❖ Operations Trail and Event Generation ❖ Assets Manipulation ❖ Ownership Control ❖ Liquidity Access

InterFi's Echelon Audit Standard

The aim of InterFi's "Echelon" standard is to analyze smart contracts and identify the vulnerabilities and the hacks. Kindly note, InterFi does not test smart contracts on testnet. It is recommended that smart contracts are thoroughly tested prior to the audit submission. Mentioned are the steps used by InterFi to audit smart contracts:

1. Solidity smart contract source code reviewal:
   - ❖ Review of the specifications, sources, and instructions provided to InterFi to make sure we understand the size, and scope of the smart contract audit.
   - ❖ Manual review of code, which is the process of reading source code line-by-line to identify potential vulnerabilities.
2. Static, Manual, and Software analysis:
   - ❖ Test coverage analysis is the process of determining whether the test cases are covering the code and how much code is exercised when we run those test cases.
   - ❖ Symbolic execution is analyzing a program to determine what inputs cause 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

Automated 3P frameworks used to assess the smart contract vulnerabilities

- ❖ Consensys Tools
- ❖ SWC Registry
- ❖ Solidity Coverage
- ❖ Open Zeppelin Code Analyzer
- ❖ Solidity Code Complier

Risk Classification

Smart contracts are generally designed to manipulate and hold funds denominated in ETH/BNB. This makes them very tempting attack targets, as a successful attack may allow the attacker to directly steal funds from the contract. Below are the typical risk levels of a smart contract:

Vulnerable: A contract is vulnerable if it has been flagged by a static analysis tool as such. As we will see later, this means that some contracts may be vulnerable because of a false positive.

Exploitable: A contract is exploitable if it is vulnerable and the vulnerability could be exploited by an external attacker. For example, if the "vulnerability" flagged by a tool is in a function that requires owning the contract, it would be vulnerable but not exploitable.

Exploited: A contract is exploited if it received a transaction on the main network which triggered one of its vulnerabilities. Therefore, a contract can be vulnerable or even exploitable without having been exploited.

Risk severity	Meaning
! High	This level vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
! Medium	This level vulnerabilities are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to high-risk severity
! Low	This level vulnerabilities should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution.
! Informational	This level vulnerabilities can be ignored. They are code style violations and informational statements in the code. They may not affect the smart contract execution

Centralization Risk

Centralization risk is the most common cause of decentralized finance hacks. When a smart contract has an active contract ownership, the risk related to centralization is elevated. There are some well-intended reasons to be an active contract owner, such as:

- Contract owner can be granted the power to `pause()` or `lock()` the contract in case of an external attack.
- Contract owner can use functions like, `include()`, and `exclude()` to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale, and to list on an exchange.

Authorizing a full centralized power to a single body can be dangerous. Unfortunately, centralization related risks are higher than common smart contract vulnerabilities. Centralization of ownership creates a risk of rug pull scams, where owners cash out tokens in such quantities that they become valueless. **Most important question to ask here is, how to mitigate centralization risk?** Here's InterFi's recommendation to lower the risks related to centralization hacks:

- Smart contract owner's private key must be carefully secured to avoid any potential hack.
- Smart contract ownership should be shared by multi-signature (multi-sig) wallets.
- Smart contract ownership can be locked in a contract, user voting, or community DAO can be introduced to unlock the ownership.

Crypto Walkers Weapons' Centralization Status

- Crypto Walkers Weapons' smart contract has an **active ownership.**
- Smart contract is **not deployed** on blockchain at the time of the audit.

Static Analysis

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable
🔒	Function is locked
🔐	Function can be accessed
!	Important functionality

ERC721	Implementation	Context, ERC165, IERC721, IERC721Metadata		
└	<Constructor>	Public !	🛑	NO!
└	supportsInterface	Public !		NO!
└	balanceOf	Public !		NO!
└	ownerOf	Public !		NO!
└	name	Public !		NO!
└	symbol	Public !		NO!
└	tokenURI	Public !		NO!
└	_baseURI	Internal 🔒		
└	approve	Public !	🛑	NO!
└	getApproved	Public !		NO!
└	setApprovalForAll	Public !	🛑	NO!
└	isApprovedForAll	Public !		NO!
└	transferFrom	Public !	🛑	NO!
└	safeTransferFrom	Public !	🛑	NO!
└	safeTransferFrom	Public !	🛑	NO!
└	_safeTransfer	Internal 🔒	🛑	
└	_exists	Internal 🔒		
└	_isApprovedOrOwner	Internal 🔒		
└	_safeMint	Internal 🔒	🛑	
└	_safeMint	Internal 🔒	🛑	
└	_mint	Internal 🔒	🛑	
└	_burn	Internal 🔒	🛑	
└	_transfer	Internal 🔒	🛑	
└	_approve	Internal 🔒	🛑	
└	_setApprovalForAll	Internal 🔒	🛑	
└	_checkOnERC721Received	Private 🔐	🛑	
└	_beforeTokenTransfer	Internal 🔒	🛑	
└	_afterTokenTransfer	Internal 🔒	🛑	
Ownable	Implementation	Context		

└	<Constructor>	Public ❗	🛑	NO❗
└	owner	Public ❗		NO❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	_transferOwnership	Internal 🔒	🛑	
MerkleProof	Library			
└	verify	Internal 🔒		
ReentrancyGuard	Implementation			
└	<Constructor>	Public ❗	🛑	NO❗
CryptoWalkersWeapons	Implementation	ERC721, Ownable, ReentrancyGuard		
└	<Constructor>	Public ❗	🛑	ERC721
└	verifyValidity	Public ❗		NO❗
└	calculateAllocation	Internal 🔒		
└	safeMint	Internal 🔒	🛑	
└	freeMint	External ❗	💵	nonReentrant
└	msMint	External ❗	💵	nonReentrant
└	adminMint	External ❗	🛑	nonReentrant onlyOwner
└	tokenURI	Public ❗		NO❗
└	baseTokenURI	Public ❗		NO❗
└	updateBaseURI	External ❗	🛑	onlyOwner
└	enableFreeMint	External ❗	🛑	onlyOwner
└	disableFreeMint	External ❗	🛑	onlyOwner
└	getSate	Public ❗		NO❗
└	setStateToAdmin	External ❗	🛑	onlyOwner
└	setStateToPublic	External ❗	🛑	onlyOwner
└	setStateToPaused	External ❗	🛑	onlyOwner
└	totalSupply	External ❗		NO❗
└	withdrawAll	External ❗	🛑	onlyOwner nonReentrant

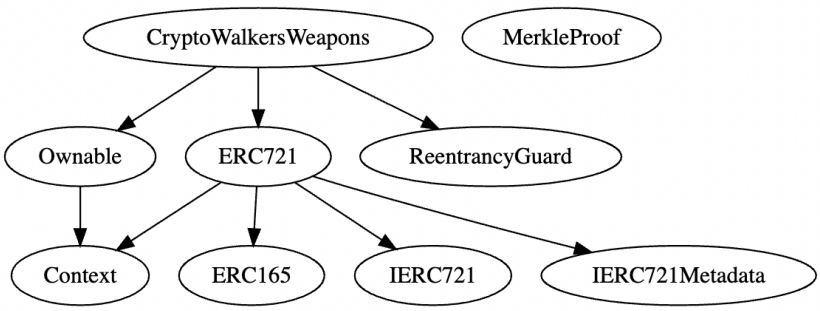
Software Analysis

Function Signatures

```
01ffc9a7  =>  supportsInterface(bytes4)
70a08231  =>  balanceOf(address)
6352211e  =>  ownerOf(uint256)
06fdde03  =>  name()
95d89b41  =>  symbol()
c87b56dd  =>  tokenURI(uint256)
743976a0  =>  _baseURI()
095ea7b3  =>  approve(address,uint256)
081812fc  =>  getApproved(uint256)
a22cb465  =>  setApprovalForAll(address,bool)
e985e9c5  =>  isApprovedForAll(address,address)
23b872dd  =>  transferFrom(address,address,uint256)
42842e0e  =>  safeTransferFrom(address,address,uint256)
b88d4fde  =>  safeTransferFrom(address,address,uint256,bytes)
24b6b8c0  =>  _safeTransfer(address,address,uint256,bytes)
f8e76cc0  =>  _exists(uint256)
4cdc9549  =>  _isApprovedOrOwner(address,uint256)
b3e1c718  =>  _safeMint(address,uint256)
6a4f832b  =>  _safeMint(address,uint256,bytes)
4e6ec247  =>  _mint(address,uint256)
9b1f9e74  =>  _burn(uint256)
30e0789e  =>  _transfer(address,address,uint256)
7b7d7225  =>  _approve(address,uint256)
8c4e3f32  =>  _setApprovalForAll(address,address,bool)
1fd01de1  =>  _checkOnERC721Received(address,address,uint256,bytes)
cad3be83  =>  _beforeTokenTransfer(address,address,uint256)
8f811a1c  =>  _afterTokenTransfer(address,address,uint256)
8da5cb5b  =>  owner()
715018a6  =>  renounceOwnership()
f2fde38b  =>  transferOwnership(address)
d29d44ee  =>  _transferOwnership(address)
5a9a49c7  =>  verify(bytes32[],bytes32,bytes32)
9a8e9747  =>  verifyValidity(address,uint256,bytes32[])
840f0720  =>  calculateAllocation(uint256)
24eeedf6  =>  safeMint(address,uint256,uint256)
e2f36dce  =>  freeMint(uint256,bytes32[])
4ec91fd0  =>  msMint(uint256,uint256,bytes32[])
c1f26123  =>  adminMint(uint256)
d547cfb7  =>  baseTokenURI()
931688cb  =>  updateBaseURI(string)
9eb39a54  =>  enableFreeMint()
adaa6c1b  =>  disableFreeMint()
f01e94a7  =>  getSate()
384b9f8a  =>  setStateToAdmin()
fe2b5a74  =>  setStateToPublic()
```

```
defc14ba  =>  setStateToPaused()
18160ddd  =>  totalSupply()
853828b6  =>  withdrawAll()
```

Inheritance Graph



Manual Analysis

Function	Description	Tested	Verdict
Total Supply	provides information about the total token supply	Yes	**Passed**
Balance Of	provides account balance of the owner's account	Yes	**Passed**
Transfer	executes transfers of a specified number of tokens to a specified address	Yes	**Passed**
Approve	allow a spender to withdraw a set number of tokens from a specified account	Yes	**Passed**
Allowance	returns a set number of tokens from a spender to the owner	Yes	**Passed**
Burn	executes transfers of a specified number of tokens to a burn address	NA	NA
NFT Mint	executes the creation of a specified number of NFTs and adds it to the total supply	Yes	**Passed**
Anti Bot	stops some or all bot wallets from interacting with the smart contract	NA	NA
Transfer Ownership	executes transfer of contract ownership to a specified wallet	Yes	**Passed**
Renounce Ownership	executes transfer of contract ownership to a dead address	Yes	**Passed**

Notable Information

- Smart contract owner can set states. In admin state, owner wallet can mint NFTs. In public state, public wallets can mint freely. Smart contract owner can enable/disable public mint.

```
function setStateToAdmin() external onlyOwner {
    activeState = State.Admin;
function setStateToPublic() external onlyOwner {
    activeState = State.Public;
function setStateToPaused() external onlyOwner {
    activeState = State.Paused;
```

- In CryptoWalkersWeapons.sol, function `getSate()` should be changed to `getState()`.
- Smart contract owner can **allow safe transfer**. This function module is used to process safe NFT wallet to wallet transfers.
- CryptoWalkersWeapons.sol smart contract uses **re-entrancy guard** to prevent re-entrant calls. Reentrancy Guard is a contract module that helps prevent reentrant calls to a function. Inheriting from Reentrancy Guard will make the nonReentrant modifier available, which can be applied to functions to make sure there are no nested (reentrant) calls to them.
- MerkleProof.sol library is established. The library with regards to CryptoWalkersWeapons.sol is verified #0x92cb4a24902597f22769dc3dd7e6ff970220e186357dc06a672fbe28c91f5ac2
- Smart contract has a **low severity issue** which may or may not create any functional vulnerability.

  "severity": 8, (! Low Severity)

  “SWC-129”

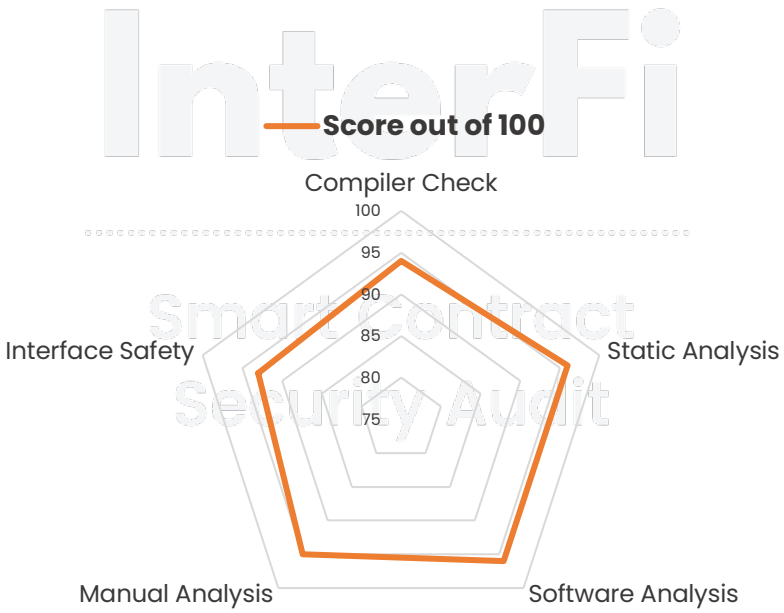
SWC Attacks

SWC ID	Description	Status
SWC-101	Integer Overflow and Underflow	**Passed**
SWC-102	Outdated Compiler Version	**! Informational**
SWC-103	Floating Pragma	**Passed**
SWC-104	Unchecked Call Return Value	**Passed**
SWC-105	Unprotected Ether Withdrawal	**Passed**
SWC-106	Unprotected SELF-DESTRUCT Instruction	**Passed**
SWC-107	Re-entrancy	**Passed**
SWC-108	State Variable Default Visibility	**Passed**
SWC-109	Uninitialized Storage Pointer	**Passed**
SWC-110	Assert Violation	**Passed**
SWC-111	Use of Deprecated Solidity Functions	**Passed**
SWC-112	Delegate Call to Untrusted Callee	**Passed**
SWC-113	DoS with Failed Call	**Passed**
SWC-114	Transaction Order Dependence	**Passed**
SWC-115	Authorization through tx.origin	**Passed**
SWC-116	Block values as a proxy for time	**Passed**
SWC-117	Signature Malleability	**Passed**
SWC-118	Incorrect Constructor Name	**Passed**

SWC-119	Shadowing State Variables	**Passed**
SWC-120	Weak Sources of Randomness from Chain Attributes	**Passed**
SWC-121	Missing Protection against Signature Replay Attacks	**Passed**
SWC-122	Lack of Proper Signature Verification	**Passed**
SWC-123	Requirement Violation	**Passed**
SWC-124	Write to Arbitrary Storage Location	**Passed**
SWC-125	Incorrect Inheritance Order	**Passed**
SWC-126	Insufficient Gas Griefing	**Passed**
SWC-127	Arbitrary Jump with Function Type Variable	**Passed**
SWC-128	DoS With Block Gas Limit	**Passed**
SWC-129	Typographical Error	**! Low**
SWC-130	Right-To-Left-Override control character (U+202E)	**Passed**
SWC-131	Presence of unused variables	**Passed**
SWC-132	Unexpected Ether balance	**Passed**
SWC-133	Hash Collisions With Multiple Variable Length Arguments	**Passed**
SWC-134	Message call with the hardcoded gas amount	**Passed**
SWC-135	Code With No Effects (Irrelevant/Dead Code)	**Passed**
SWC-136	Unencrypted Private Data On-Chain	**Passed**

Risk Status & Radar Chart

Risk Severity	Status
High	No high severity issues identified
Medium	No medium severity issues identified
Low	1 low severity issue identified
Informational	1 informational severity issue identified
Centralization Risk	Active contract ownership identified



Auditor's Verdict

InterFi team has performed a line-by-line manual analysis and automated review of smart contracts. Smart contracts were analyzed mainly for common contract vulnerabilities, exploits, and manipulation hacks. According to the audit:

- Crypto Walkers Weapons' smart contract source code has **LOW RISK SEVERITY**
- Crypto Walkers Weapons' smart contract has an **ACTIVE OWNERSHIP**
- Crypto Walkers Weapons' centralization risk correlated to the active owner is **LOW**

Note for stakeholders

- Be aware that active smart contract owner privileges constitute an elevated impact on smart contract safety and security.
- If the smart contract is not deployed on any blockchain at the time of the audit, the contract can be modified or altered before blockchain development. Verify contract's deployment status in the audit report.
- Make sure that the project team's KYC/identity is verified by an independent firm.
- Always check if the contract's liquidity is locked. A longer liquidity lock plays an important role in the project's longevity. It is recommended to have multiple liquidity providers.
- Examine the unlocked token supply in the owner, developer, or team's private wallets. Understand the project's tokenomics, and make sure the tokens outside of the LP Pair are vested or locked for a longer period.

Important Disclaimer

InterFi Network provides contract development, testing, auditing and project evaluation services for blockchain projects. The purpose of the audit is to analyze the on-chain smart contract source code and to provide a basic overview of the project. **This report should not be transmitted, disclosed, referred to, or relied upon by any person for any purpose without InterFi's prior written consent.**

InterFi provides the easy-to-understand assessment of the project, and the smart contract (otherwise known as the source code). The audit makes no statements or warranties on the security of the code. It also cannot be considered as enough assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have used all the data at our disposal to provide the transparent analysis, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. **Be aware that smart contracts deployed on a blockchain aren't resistant to external vulnerability, or a hack. Be aware that active smart contract owner privileges constitute an elevated impact on smart contract safety and security. Therefore, InterFi does not guarantee the explicit security of the audited smart contract.**

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

This report should not be considered as an endorsement or disapproval of any project or team. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. Do conduct your due diligence and consult your financial advisor before making any investment decisions.

About InterFi Network

InterFi Network provides intelligent blockchain solutions. InterFi is developing an ecosystem that is seamless and responsive. Some of our services: Blockchain Security, Token Launchpad, NFT Marketplace, etc. **InterFi's mission is to interconnect multiple services like Blockchain Security, DeFi, Gaming, and Marketplace under one ecosystem that is seamless, multi-chain compatible, scalable, secure, fast, responsive, and easy to use.**

InterFi is built by a decentralized team of UI experts, contributors, engineers, and enthusiasts from all over the world. Our team currently consists of 6+ core team members, and 10+ casual contributors. **InterFi provides manual, static, and automatic smart contract analysis, to ensure that project is checked against known attacks and potential vulnerabilities.**

To learn more, visit **https://interfi.network**

To view our audit portfolio, visit **https://github.com/interfinetwork**

To book an audit, message **https://t.me/interfiaudits**

@INTERFINETWORK

RELENTLESSLY SECURING THE PUBLIC BLOCKCHAIN | MADE IN CANADA