



SMART CONTRACT SECURITY AUDIT OF THE CRONOS STEAKHOUSE



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	The Cronos Steakhouse
Website	https://www.cronossteakhouse.com/
Telegram	https://t.me/CronosSteakhouse/
Twitter	https://twitter.com/CronoSteakHouse/
Discord	https://discord.gg/fpTgAReDyN/
Report Date	June 12, 2022

About The Cronos Steakhouse

Originally started as a decentralised miner fork of Baked Beans, The Cronos Steakhouse has evolved into 5 Grills with much more to come with aims to help lock tokens out of the ecosystem to help enable a healthier price point. Custom features such as Lottery and NFT Discounts with giveaways and more centred around the #crofam



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:

- ❖ The Cronos Steakhouse's solidity source codes have **LOW RISK SEVERITY**
- ❖ The Cronos Steakhouse's smart contracts have **ACTIVE OWNERSHIP**
- ❖ Centralization risk correlated to the active owner is **MEDIUM**
- ❖ Important privileges: **ADD OR REMOVE DISCOUNT TOKEN, SEED MARKET, GRILL / EAT / RE-GRILL STEAK**

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.

- 🔴 Steakhouse contract: **0xd3c053dE4fF3d3fA734fCc74f7269c1227C170df**
- 🔴 MMF Steakhouse contract: **0xE56Bb7b50FA602c42C90Cd0162e21D45aaDe61a4**
- 🔴 SVN Steakhouse contract: **0x240b039f58C8516e670602b058aDc395dC48DdfD**
- 🔴 USDC Steakhouse contract: **0x64468fa3b842082eBC844Bd74D16FEb093a12C45**
- 🔴 Lottery Steakhouse contract: **0x33ca6054Ad5449c8add78d83E8a095a2E1C13FCf**
- 🔗 Blockchain: **Cronos Chain**
- ✅ 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.....	13
Manual Analysis.....	15
SWC Attacks	18
Risk Status & Radar Chart.....	20

Audit Summary

Auditor's Verdict	21
-------------------------	----

Legal Advisory

Important Disclaimer	22
About InterFi Network	23



Audit Scope

InterFi was consulted by The Cronos Steakhouse 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:

- ❖ Steakhouse.sol
- ❖ MMFSteakhouse.sol
- ❖ SVNSteakhouse.sol
- ❖ USDCSteakhouse.sol
- ❖ CROLotterySteakhouse.sol

Solidity Source Codes On Blockchain (Verified Contract Source Codes)

<https://cronoscan.com/address/0xd3c053dE4fF3d3fA734fCc74f7269c1227C170df#code>

<https://cronoscan.com/address/0xE56Bb7b50FA602c42C90Cd0162e21D45aaDe61a4#code>

<https://cronoscan.com/address/0x240b039f58C8516e670602b058aDc395dC48DdfD#code>

<https://cronoscan.com/address/0x64468fa3b842082eBC844Bd74D16FEb093a12C45#code>

<https://cronoscan.com/address/0x33ca6054Ad5449c8add78d83E8a095a2E1C13FCf#code>

SHA-1 Hash

Solidity source codes are audited at hash #f28b5810d17493dd300ac96a65199dda3df869e5



Audit Methodology

The scope of this report is to audit smart contract source codes of The Cronos Steakhouse. InterFi has scanned contracts and reviewed codes for common vulnerabilities, exploits, hacks, and backdoors. 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, exploits, and hacks:

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

❖ Deployment Consistency

❖ Repository Consistency

❖ Data Consistency

❖ Token Supply Manipulation

❖ Access Control and Authorization

❖ Operations Trail and Event Generation

❖ Assets Manipulation

❖ Ownership Control

❖ Liquidity Access

Source Code Review



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 Compiler



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.



Smart Contract Security Audit

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.

Centralization Status

















- ❖ The Cronos Steakhouse's smart contracts have **active ownership**.
- ❖ Smart contract ownership is set to **0xcbd3e8b401f659c4037074cf2946f359de1c12e4** 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

```

| **Context** | Implementation | |||
| L | _msgSender | Internal  | | |
| L | _msgData | Internal  | | |
| |||||
| **Ownable** | Implementation | Context |||
| L | <Constructor> | Public ! |  | NO ! |
| L | owner | Public ! | | NO ! |
| L | renounceOwnership | Public ! |  | onlyOwner |
| L | transferOwnership | Public ! |  | onlyOwner |
| L | _transferOwnership | Internal  |  | |
| |||||
| **Steakhouse** | Implementation | Ownable |||
| L | <Constructor> | Public ! |  | NO ! |
| L | reGrill | Public ! |  | NO ! |
| L | eatSteak | Public ! |  | NO ! |
| L | beanRewards | Public ! | | NO ! |
| L | grillSteak | Public ! |  | NO ! |
| L | calculateTrade | Private  | | |
| L | calculateSteakSell | Public ! | | NO ! |
| L | calculateSteakBuy | Public ! | | NO ! |
| L | calculateSteakBuySimple | Public ! | | NO ! |
| L | devFee | Private  | | |
| L | seedMarket | Public ! |  | onlyOwner |
| L | getBalance | Public ! | | NO ! |
| L | getMyMiners | Public ! | | NO ! |
| L | getMySteak | Public ! | | NO ! |
| L | getSteakSinceLastHatch | Public ! | | NO ! |
| L | min | Private  | | |
| |||||
| **MFSSteakhouse** | Implementation | Ownable |||
| L | <Constructor> | Public ! |  | NO ! |

```



```

| L | reGrill | Public ! | ● | NO ! |
| L | eatSteak | Public ! | ● | NO ! |
| L | steakRewards | Public ! | | NO ! |
| L | grillSteak | Public ! | ● | NO ! |
| L | calculateTrade | Private 🗝️ | | |
| L | calculateSteakSell | Public ! | | NO ! |
| L | calculateSteakBuy | Public ! | | NO ! |
| L | calculateSteakBuySimple | Public ! | | NO ! |
| L | devFee | Private 🗝️ | | |
| L | getDevFee | Public ! | | NO ! |
| L | burnFee | Private 🗝️ | | |
| L | seedMarket | Public ! | ● | onlyOwner |
| L | getBalance | Public ! | | NO ! |
| L | getMyMiners | Public ! | | NO ! |
| L | getMySteak | Public ! | | NO ! |
| L | getSteakSinceLastHatch | Public ! | | NO ! |
| L | min | Private 🗝️ | | |
| L | addOrUpdateDiscountToken | External ! | ● | onlyOwner |
| L | removeDiscountToken | External ! | ● | onlyOwner |
| L | getInvestorCount | External ! | | NO ! |
|||||
| **SVNSteakhouse** | Implementation | Ownable |||
| L | <Constructor> | Public ! | ● | NO ! |
| L | reGrill | Public ! | ● | NO ! |
| L | eatSteak | Public ! | ● | NO ! |
| L | steakRewards | Public ! | | NO ! |
| L | grillSteak | Public ! | ● | NO ! |
| L | calculateTrade | Private 🗝️ | | |
| L | calculateSteakSell | Public ! | | NO ! |
| L | calculateSteakBuy | Public ! | | NO ! |
| L | calculateSteakBuySimple | Public ! | | NO ! |
| L | devFee | Private 🗝️ | | |
| L | getDevFee | Public ! | | NO ! |
| L | burnFee | Private 🗝️ | | |
| L | seedMarket | Public ! | ● | onlyOwner |
| L | getBalance | Public ! | | NO ! |
| L | getMyMiners | Public ! | | NO ! |
| L | getMySteak | Public ! | | NO ! |
| L | getSteakSinceLastHatch | Public ! | | NO ! |
| L | min | Private 🗝️ | | |
| L | addOrUpdateDiscountToken | External ! | ● | onlyOwner |
| L | removeDiscountToken | External ! | ● | onlyOwner |
| L | getInvestorCount | External ! | | NO ! |
|||||
| **USDCSteakhouse** | Implementation | Ownable |||
| L | <Constructor> | Public ! | ● | NO ! |
| L | reGrill | Public ! | ● | NO ! |
| L | eatSteak | Public ! | ● | NO ! |
| L | steakRewards | Public ! | | NO ! |
| L | grillSteak | Public ! | ● | NO ! |

```



```

| L | calculateTrade | Private 🔒 | | |
| L | calculateSteakSell | Public ! | | NO ! |
| L | calculateSteakBuy | Public ! | | NO ! |
| L | calculateSteakBuySimple | Public ! | | NO ! |
| L | devFee | Private 🔒 | | |
| L | getDevFee | Public ! | | NO ! |
| L | burnFee | Private 🔒 | | |
| L | seedMarket | Public ! | 🔴 | onlyOwner |
| L | getBalance | Public ! | | NO ! |
| L | getMyMiners | Public ! | | NO ! |
| L | getMySteak | Public ! | | NO ! |
| L | getSteakSinceLastHatch | Public ! | | NO ! |
| L | min | Private 🔒 | | |
| L | addOrUpdateDiscountToken | External ! | 🔴 | onlyOwner |
| L | removeDiscountToken | External ! | 🔴 | onlyOwner |
| L | getInvestorCount | External ! | | NO ! |
|||||
| **CR0LotterySteakhouse** | Implementation | Ownable |||
| L | <Constructor> | Public ! | 🔴 | NO ! |
| L | reGrill | Public ! | 🔴 | NO ! |
| L | eatSteak | Public ! | 🔴 | NO ! |
| L | steakRewards | Public ! | | NO ! |
| L | grillSteak | Public ! | 🍷 | NO ! |
| L | calculateTrade | Private 🔒 | | |
| L | calculateSteakSell | Public ! | | NO ! |
| L | calculateSteakBuy | Public ! | | NO ! |
| L | calculateSteakBuySimple | Public ! | | NO ! |
| L | devFee | Private 🔒 | | |
| L | getDevFee | Public ! | | NO ! |
| L | marketingFee | Private 🔒 | | |
| L | seedMarket | Public ! | 🍷 | onlyOwner |
| L | getBalance | Public ! | | NO ! |
| L | getMyChefs | Public ! | | NO ! |
| L | getMySteak | Public ! | | NO ! |
| L | getSteakSinceLastGrill | Public ! | | NO ! |
| L | min | Private 🔒 | | |
| L | addOrUpdateDiscountToken | External ! | 🔴 | onlyOwner |
| L | removeDiscountToken | External ! | 🔴 | onlyOwner |
| L | getInvestorCount | External ! | | NO ! |
| L | _getRand | Internal 🔒 | | |
| L | rewardWinner | Internal 🔒 | 🔴 | |

```



Software Analysis

Function Signatures

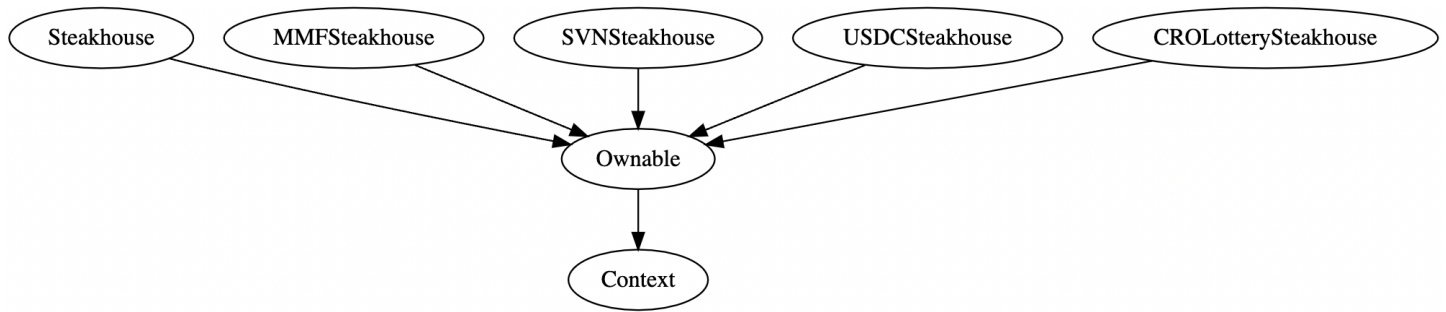
```

119df25f => _msgSender()
8b49d47e => _msgData()
8da5cb5b => owner()
715018a6 => renounceOwnership()
f2fde38b => transferOwnership(address)
d29d44ee => _transferOwnership(address)
818f1abf => reGrill(address)
5ec737bd => eatSteak()
a507abee => beanRewards(address)
d1fa8864 => grillSteak(address)
229824c4 => calculateTrade(uint256,uint256,uint256)
4051273d => calculateSteakSell(uint256)
82c5d57c => calculateSteakBuy(uint256,uint256)
d796e734 => calculateSteakBuySimple(uint256)
3bc0461a => devFee(uint256)
3c5f07cb => seedMarket()
12065fe0 => getBalance()
4b634b06 => getMyMiners(address)
e02546e8 => getMySteak(address)
a4db36da => getSteakSinceLastHatch(address)
7ae2b5c7 => min(uint256,uint256)
aa76882e => steakRewards(address)
8b568968 => grillSteak(address,uint256)
c7287e9d => getDevFee()
49ae028a => burnFee(uint256)
3b653755 => seedMarket(uint256)
ba39e433 => addOrUpdateDiscountToken(address,uint256,uint256,uint256)
69eea55a => removeDiscountToken(address)
960524e3 => getInvestorCount()
82fb7119 => marketingFee(uint256)
dd477dd9 => getMyChefs(address)
f441a156 => getSteakSinceLastGrill(address)
d0ef1048 => _getRand()
373c7143 => rewardWinner(address)

```



Inheritance Graph



InterFi

Smart Contract
Security Audit



Manual Analysis

Function	Description	Available	Status
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
Reward	executes transfers of a specified reward token to a specified address	Yes	Passed
Contract Fees	executes fee collection from swap events and/or transfer events	Yes	Passed
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 contracts **do not utilize re-entrancy guard** to prevent re-entrant calls. However, the functions may not be vulnerable to re-entrancy as transfers are done using transfer. The amount of gas passed to transfer CRO is so little that wouldn't ever be enough to start an execution flow.
- ❖ MMFSteakhouse.sol mining token: **0x97749c9B61F878a880DfE312d2594AE07AEd7656**

```
uint256 private STEAK_TO_HATCH_1cheffs = 1080000;
uint256 private devFeeVal = 2;
uint256 private burnFeeVal = 2;
IERC20 private miningToken = IERC20(0x97749c9B61F878a880DfE312d2594AE07AEd7656);
uint256 public WITHDRAW_COOLDOWN = 6 days;
```

- ❖ SVNSteakhouse.sol mining token: **0x654bAc3eC77d6dB497892478f854cF6e8245DcA9**

```
uint256 private STEAK_TO_HATCH_1cheffs = 1080000;
uint256 private devFeeVal = 2;
uint256 private burnFeeVal = 2;
IERC20 private miningToken = IERC20(0x654bAc3eC77d6dB497892478f854cF6e8245DcA9);
uint256 public WITHDRAW_COOLDOWN = 6 days;
```

- ❖ USDCSteakhouse.sol mining token: **0xc21223249CA28397B4B6541dfFaEcC539BfF0c59**

```
uint256 private STEAK_TO_HATCH_1cheffs = 1080000;
uint256 private devFeeVal = 2;
uint256 private marketFeeVal = 1;
IERC20 private miningToken = IERC20(0xc21223249CA28397B4B6541dfFaEcC539BfF0c59);
uint256 public WITHDRAW_COOLDOWN = 6 days;
```

- ❖ Smart contracts utilize `grillSteak()`, `reGrill()` and `eatSteak()`.

```
function grillSteak(address ref) public payable {
    require(initialized);
    uint256 contractBalance = address(this).balance;
    uint256 meatBought = calculateSteakBuy(msg.value, contractBalance);
    meatBought = meatBought - devFee(meatBought) - marketingFee(meatBought);
    function reGrill(address ref) public {
        require(initialized);
    }
    function eatSteak() public {
```




```
require(initialized);
require(lastSell[msg.sender] + WITHDRAW_COOLDOWN <= block.timestamp, "You can't withdraw
for a while");
```

- ❖ Smart contracts can **seed market** after initialization.

```
function seedMarket() public payable onlyOwner {
    require(marketSteak == 0);
    initialized = true;
```

- ❖ Smart contracts call `receive()`, it is executed on a call to the contract with empty call data. This is the function that is executed on plain ether transfers such as `send()`, and `transfer()`. Make sure the contract can receive token through a regular transaction, and does not throw an exception.

```
receive() external payable {}
```

- ❖ Smart contracts utilize **redundant code** for `transferOwnership()`. Ideal transfer ownership code should look be written like:

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
```

- ❖ Smart contracts charge `devFee()`, `marketFee()` and `burnFee()`.
- ❖ Smart contract has an **informational severity issue** which may or may not create any functional vulnerability.

"severity": 8, (! Low Severity)

"Use of block.timestamp"



SWC Attacks

SWC ID	Description	Status
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	! Low
SWC-103	Floating Pragma	! Informational
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELF-DESTRUCT Instruction	Passed
SWC-107	Re-entrancy	! Low
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

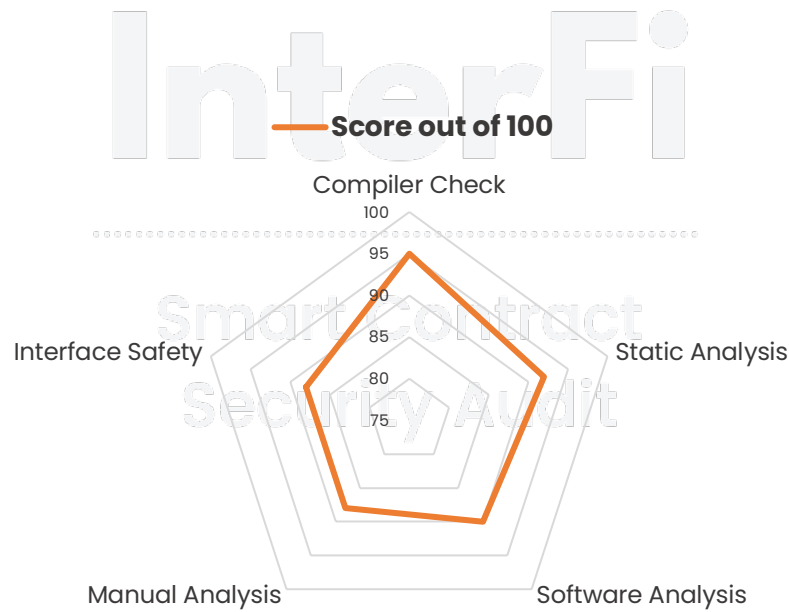


SWC-119	Shadowing State Variables	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed
SWC-122	Lack of Proper Signature Verification	Passed
SWC-123	Requirement Violation	Passed
SWC-124	Write to Arbitrary Storage Location	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	Passed
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)	! Informational
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	2 low severity issues identified
Informational	3 informational severity issues identified
Centralization Risk	Active 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:

- ❖ The Cronos Steakhouse's solidity source codes have **LOW RISK SEVERITY**
- ❖ The Cronos Steakhouse's smart contracts have **ACTIVE OWNERSHIP**
- ❖ Centralization risk correlated to the active owner is **MEDIUM**

InterFi

.....

Note for stakeholders

Smart Contract

Security Audit

- ❖ 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>





RELENTLESSLY SECURING THE PUBLIC BLOCKCHAIN | MADE IN CANADA 