InterFi NETWORK

@INTERFINETWORK

SMART CONTRACT SECURITY AUDIT OF

KawaFarm 2.0

(Kawakami's Staking Contract)

SMART CONTRACT AUDIT | TEAM KYC | PROJECT EVALUATION

RELENTLESSLY SECURING THE PUBLIC BLOCKCHAIN | MADE IN CANADA

Summary

Auditing Firm	InterFi Network
Architecture	InterFi "Echelon" Auditing Standard
Smart Contract Audit Approved By	Chris | Blockchain Specialist at InterFi Network
Platform	Solidity
Audit Check (Mandatory)	Static, Software, Auto Intelligent & Manual Analysis
Consultation Request Date	October 17, 2021
Report Date	October 23, 2021

Audit Summary

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

- **KawaFarm's smart contract source codes have LOW RISK SEVERITY.**
- **KawaFarm has PASSED the smart contract audit.**
- **KawaFarm is a part of Kawakami Ecosystem. Learn more at: https://kawatoken.io/**

KawaFarm is Kawakami's staking contract. This report is only for KawaFarm's smart contract. If you're looking for Kawakami's smart contract audit, please visit:

https://kawatoken.io/wp-content/uploads/2021/10/Kawakami_AuditReport_InterFi.pdf

For the detailed understanding of risk severity, source code vulnerability, and functional test, kindly refer to the audit.

Table Of Contents

Project Overview

InterFi was consulted by Kawakami on October 20, 2021 to conduct a smart contract security audit of their staking app (KawaFarm) source code.

KawaFarm 2.0: Farm $xKAWA by staking the most popular community tokens such as SHIB, SAITAMA, FLOKI and more! How the rewards yield work on KawaFarm 2.0?

2.5% of the total tokens allocated to a pool are released per week and distributed to stakers based on their share rate in a pool. Each pool has a release rate of 2.5% tokens per week. At this rate, the farm will run for exactly 40 weeks. As KawaFarm get closer to the depletion of farm tokens, we will announce a strategy on how to keep it going.

Project	KawaFarm 2.0
Blockchain	Ethereum Chain
Language	Solidity
Contract	0xC68844Cd3BA9d3Ad88F2cC278213F64b8C0bCddf
Website	https://farm.kawatoken.io/
Telegram	https://t.me/kawatoken
Twitter	https://twitter.com/Kawakami_Inu

Public logo

Solidity Source Code On Blockchain (EtherScan Verified Contract Source Code)

https://etherscan.io/address/0xc68844cd3ba9d3ad88f2cc278213f64b8c0bcddf#code

Contract Name: KawaFarm (TransparentUpgradeableProxy)

Compiler Version: v0.8.2+commit.661d1103

Optimization Enabled: Yes with 200 runs

Solidity Source Code On InterFi GitHub

https://github.com/interfinetwork/audited-codes/blob/main/KawaFarm.sol

GitHub Commits

Solidity source code committed at: c320dbd563f472d0c32ebc0c1029acea5cf5efb7

Solidity Codes Under Scope (Solidity **Standard Json-Input** format)

- import.sol
- ERC1967Proxy.sol
- TransparentUpgradeableProxy.sol
- ProxyAdmin.sol
- Proxy.sol
- ERC1967Upgrade.sol
- IBeacon.sol
- Address.sol
- StorageSlot.sol
- Ownable.sol
- Context.sol
- UUPSUpgradeable.sol
- Proxiable.sol

Audit Scope & Methodology

The scope of this report is to audit the smart contract source code of KawaFarm. The source code can be viewed in its entirety on

https://etherscan.io/address/0xc68844cd3ba9d3ad88f2cc278213f64b8c0bcddf#code

InterFi has scanned the contract and reviewed the project for common vulnerabilities, exploits, hacks, and back-doors. Below is the list of commonly known smart contract vulnerabilities, exploits, and hacks:

Category	
Smart Contract Vulnerabilities	❖ Re-entrancy (RE) ❖ Unhandled Exceptions (UE) ❖ Transaction Order Dependency (TO) ❖ Integer Overflow (IO) ❖ Unrestricted Action (UA) ❖ Ownership Takeover ❖ Gas Limit and Loops
Source Code Review	❖ Deployment Consistency ❖ Repository Consistency ❖ Data Consistency ❖ Token Supply Manipulation
Functional Assessment	❖ Access Control and Authorization ❖ Operations Trail and Event Generation ❖ Assets Manipulation ❖ Liquidity Access

InterFi's Echelon Audit Standard

The aim of InterFi's "Echelon" standard is to analyze the smart contract and identify the vulnerabilities and the hacks in the smart contract. Mentioned are the steps used by ECHELON-1 to assess the smart contract:

1. Solidity smart contract source code reviewal:
   - ❖ Review of the specifications, sources, and instructions provided to InterFi to make sure we understand the size, scope, and functionality of the smart contract.
   - ❖ Manual review of code, which is the process of reading source code line-byline to identify potential vulnerabilities.
2. Static, Manual, and Automated AI analysis:
   - ❖ Test coverage analysis, which 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, 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

Automated 3P frameworks used to assess the smart contract vulnerabilities

- ❖ Slither
- ❖ Consensys MythX
- ❖ Consensys Surya
- ❖ Open Zeppelin Code Analyzer
- ❖ Solidity Code Complier

InterFi's 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 which requires to own 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
! Critical	This level vulnerabilities could be exploited easily, and can lead to asset loss, data loss, asset manipulation, or data manipulation. They should be fixed right away.
! High	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 critical risk severity
! Medium	This level vulnerabilities are should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution.
! Low	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

Smart Contract – Overview

Contract information

Query	Result
Name	TransparentUpgradeableProxy
Dev Address	0xd0b84202bd02eb1b9f62b30de1d8985f65d8dfb3
Owner	0xc0e9f051c6ef76d15f777ae8a7cb14c03550e79e
Reward Token	0xdc386452f9ffda7f0d2940e5c60dc0f9469b1097
Uniswap V2 Router	0x7a250d5630b4cf539739df2c5dacb4c659f2488d

Smart Contract – Static Analysis

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

AdminUpgradeabilityProxy	Implementation	TransparentUpgradeableProxy		
└	<Constructor>	Public !	💵	TransparentUpgradeableProxy
ERC1967Proxy	Implementation	Proxy, ERC1967Upgrade		
└	<Constructor>	Public !	💵	NO!
└	_implementation	Internal 🔒		
TransparentUpgradeableProxy	Implementation	ERC1967Proxy		
└	<Constructor>	Public !	💵	ERC1967Proxy
└	admin	External !	🛑	ifAdmin
└	implementation	External !	🛑	ifAdmin
└	changeAdmin	External !	🛑	ifAdmin
└	upgradeTo	External !	🛑	ifAdmin
└	upgradeToAndCall	External !	💵	ifAdmin
└	_admin	Internal 🔒		
└	_beforeFallback	Internal 🔒	🛑	
ProxyAdmin	Implementation	Ownable		
└	getProxyImplementation	Public !		NO!
└	getProxyAdmin	Public !		NO!
└	changeProxyAdmin	Public !	🛑	onlyOwner
└	upgrade	Public !	🛑	onlyOwner
└	upgradeAndCall	Public !	💵	onlyOwner
Proxy	Implementation			
└	_delegate	Internal 🔒	🛑	
└	_implementation	Internal 🔒		
└	_fallback	Internal 🔒	🛑	
└	<Fallback>	External !	💵	NO!
└	<Receive Ether>	External !	💵	NO!
└	_beforeFallback	Internal 🔒	🛑	

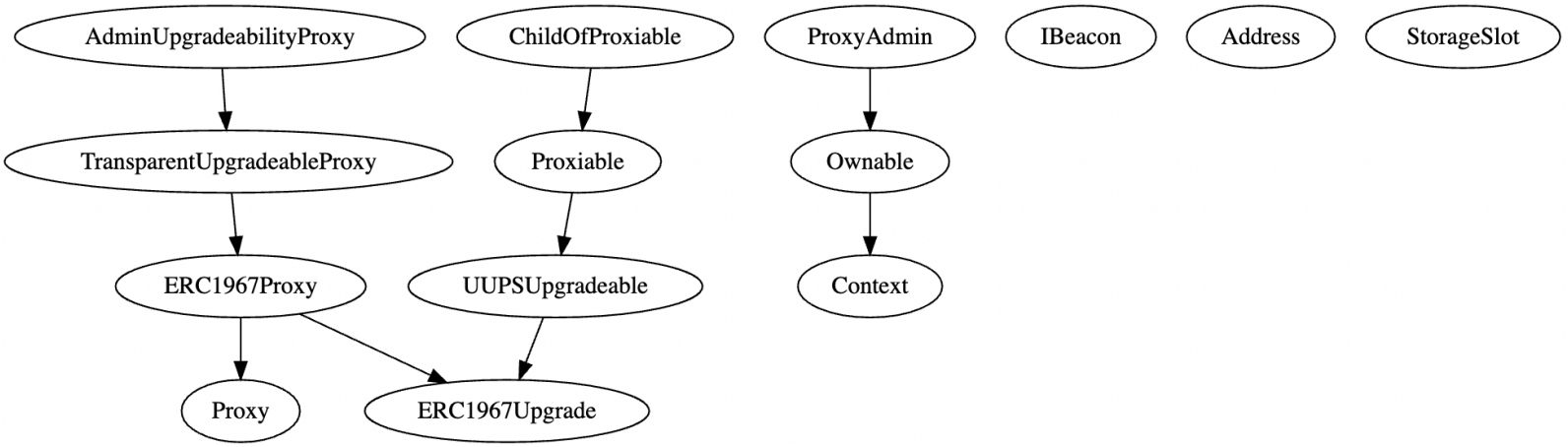
```
||||||
| **ERC1967Upgrade** | Implementation |  |||
| └ | _getImplementation | Internal 🔒 |  | |
| └ | _setImplementation | Private 🔐 | 🛑 | |
| └ | _upgradeTo | Internal 🔒 | 🛑 | |
| └ | _upgradeToAndCall | Internal 🔒 | 🛑 | |
| └ | _upgradeToAndCallSecure | Internal 🔒 | 🛑 | |
| └ | _upgradeBeaconToAndCall | Internal 🔒 | 🛑 | |
| └ | _getAdmin | Internal 🔒 |  | |
| └ | _setAdmin | Private 🔐 | 🛑 | |
| └ | _changeAdmin | Internal 🔒 | 🛑 | |
| └ | _getBeacon | Internal 🔒 |  | |
| └ | _setBeacon | Private 🔐 | 🛑 | |
||||||
| **IBeacon** | Interface |  |||
| └ | implementation | External ❗ |  |NO❗ |
||||||
| **Address** | Library |  |||
| └ | isContract | Internal 🔒 |  | |
| └ | sendValue | Internal 🔒 | 🛑 | |
| └ | functionCall | Internal 🔒 | 🛑 | |
| └ | functionCall | Internal 🔒 | 🛑 | |
| └ | functionCallWithValue | Internal 🔒 | 🛑 | |
| └ | functionCallWithValue | Internal 🔒 | 🛑 | |
| └ | functionStaticCall | Internal 🔒 |  | |
| └ | functionStaticCall | Internal 🔒 |  | |
| └ | functionDelegateCall | Internal 🔒 | 🛑 | |
| └ | functionDelegateCall | Internal 🔒 | 🛑 | |
||||||
| **Ownable** | Implementation | Context |||
| └ | <Constructor> | Public ❗ | 🛑 |NO❗ |
| └ | owner | Public ❗ |  |NO❗ |
| └ | renounceOwnership | Public ❗ | 🛑 | onlyOwner |
| └ | transferOwnership | Public ❗ | 🛑 | onlyOwner |
||||||
| **Context** | Implementation |  |||
| └ | _msgSender | Internal 🔒 |  | |
| └ | _msgData | Internal 🔒 |  | |
||||||
| **UUPSUpgradeable** | Implementation | ERC1967Upgrade |||
| └ | upgradeTo | External ❗ | 🛑 |NO❗ |
| └ | upgradeToAndCall | External ❗ | 💵 |NO❗ |
| └ | _authorizeUpgrade | Internal 🔒 | 🛑 | |
||||||
| **Proxiable** | Implementation | UUPSUpgradeable |||
| └ | _authorizeUpgrade | Internal 🔒 | 🛑 | |
| └ | _beforeUpgrade | Internal 🔒 | 🛑 | |
||||||
| **ChildOfProxiable** | Implementation | Proxiable |||
| └ | _beforeUpgrade | Internal 🔒 | 🛑 | |
```

Smart Contract – Software Analysis

Function Signatures

```
16279055  =>  isContract(address)
34140748  =>  _upgradeTo(address)
59679b0f  =>  _implementation()
f851a440  =>  admin()
5c60da1b  =>  implementation()
8f283970  =>  changeAdmin(address)
3659cfe6  =>  upgradeTo(address)
4f1ef286  =>  upgradeToAndCall(address,bytes)
01bc45c9  =>  _admin()
50c727ad  =>  _beforeFallback()
bb619d82  =>  getProxyImplementation(TransparentUpgradeableProxy)
204865fd  =>  getProxyAdmin(TransparentUpgradeableProxy)
b0cc3554  =>  changeProxyAdmin(TransparentUpgradeableProxy,address)
055543f8  =>  upgrade(TransparentUpgradeableProxy,address)
99bff37b  =>  upgradeAndCall(TransparentUpgradeableProxy,address,bytes)
f13101e9  =>  _delegate(address)
af8f35c4  =>  _fallback()
42404e07  =>  _getImplementation()
bb913f41  =>  _setImplementation(address)
267b04ae  =>  _upgradeToAndCall(address,bytes,bool)
ce2ea66a  =>  _upgradeToAndCallSecure(address,bytes,bool)
9ba186fe  =>  _upgradeBeaconToAndCall(address,bytes,bool)
839f5fb8  =>  _getAdmin()
3a74a767  =>  _setAdmin(address)
353dfc01  =>  _changeAdmin(address)
2bad8ba0  =>  _getBeacon()
073d36b4  =>  _setBeacon(address)
24a084df  =>  sendValue(address,uint256)
a0b5ffb0  =>  functionCall(address,bytes)
241b5886  =>  functionCall(address,bytes,string)
2a011594  =>  functionCallWithValue(address,bytes,uint256)
d525ab8a  =>  functionCallWithValue(address,bytes,uint256,string)
18c2c6a2  =>  _verifyCallResult(bool,bytes,string)
e8ff0b1d  =>  getAddressSlot(bytes32)
37f4a46d  =>  getBooleanSlot(bytes32)
dd4e2762  =>  getBytes32Slot(bytes32)
949a352c  =>  getUint256Slot(bytes32)
8da5cb5b  =>  owner()
715018a6  =>  renounceOwnership()
f2fde38b  =>  transferOwnership(address)
119df25f  =>  _msgSender()
8b49d47e  =>  _msgData()
5ec29272  =>  _authorizeUpgrade(address)
24ff6f63  =>  _beforeUpgrade(address)
```

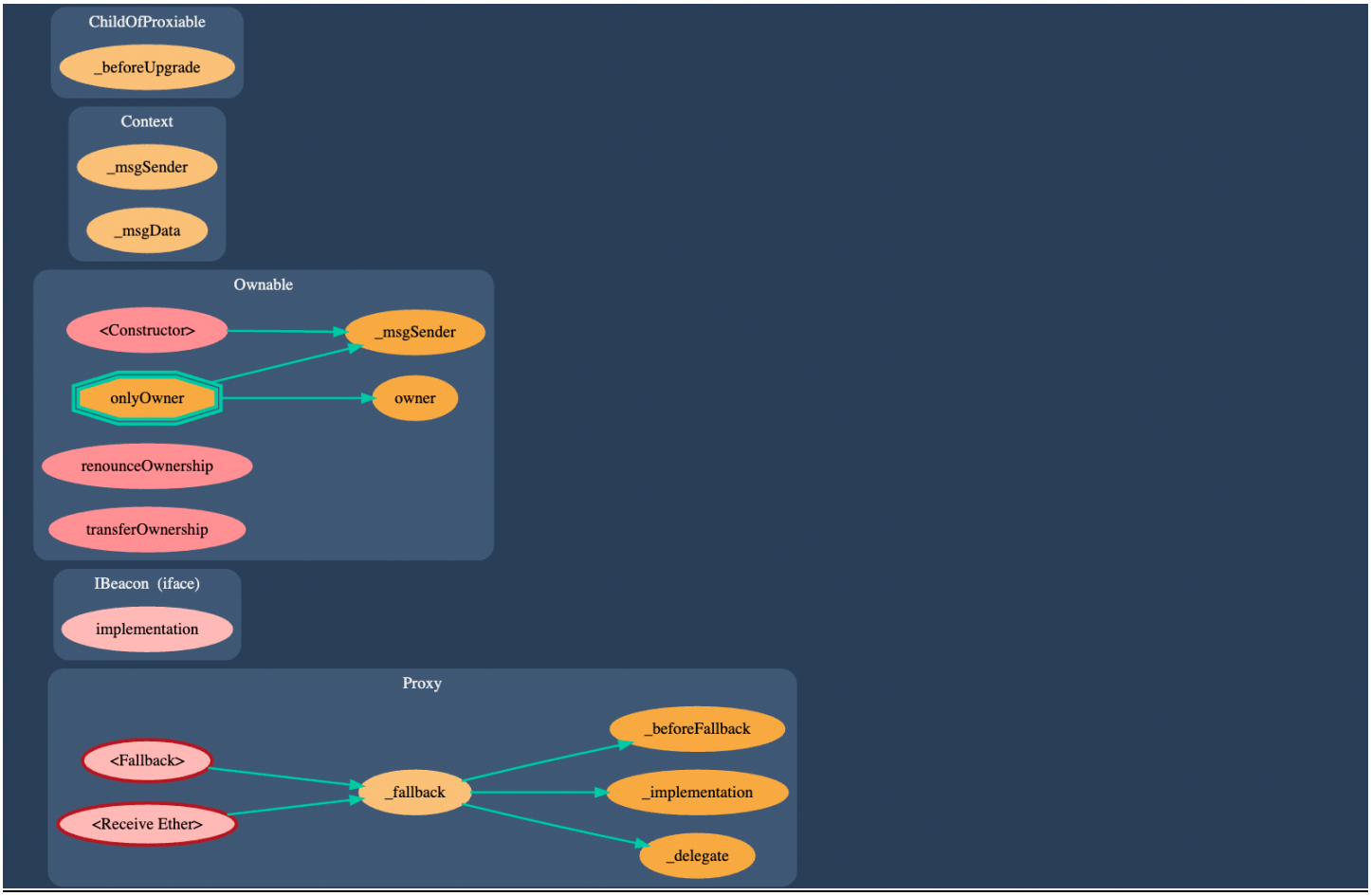
Inheritance Graph



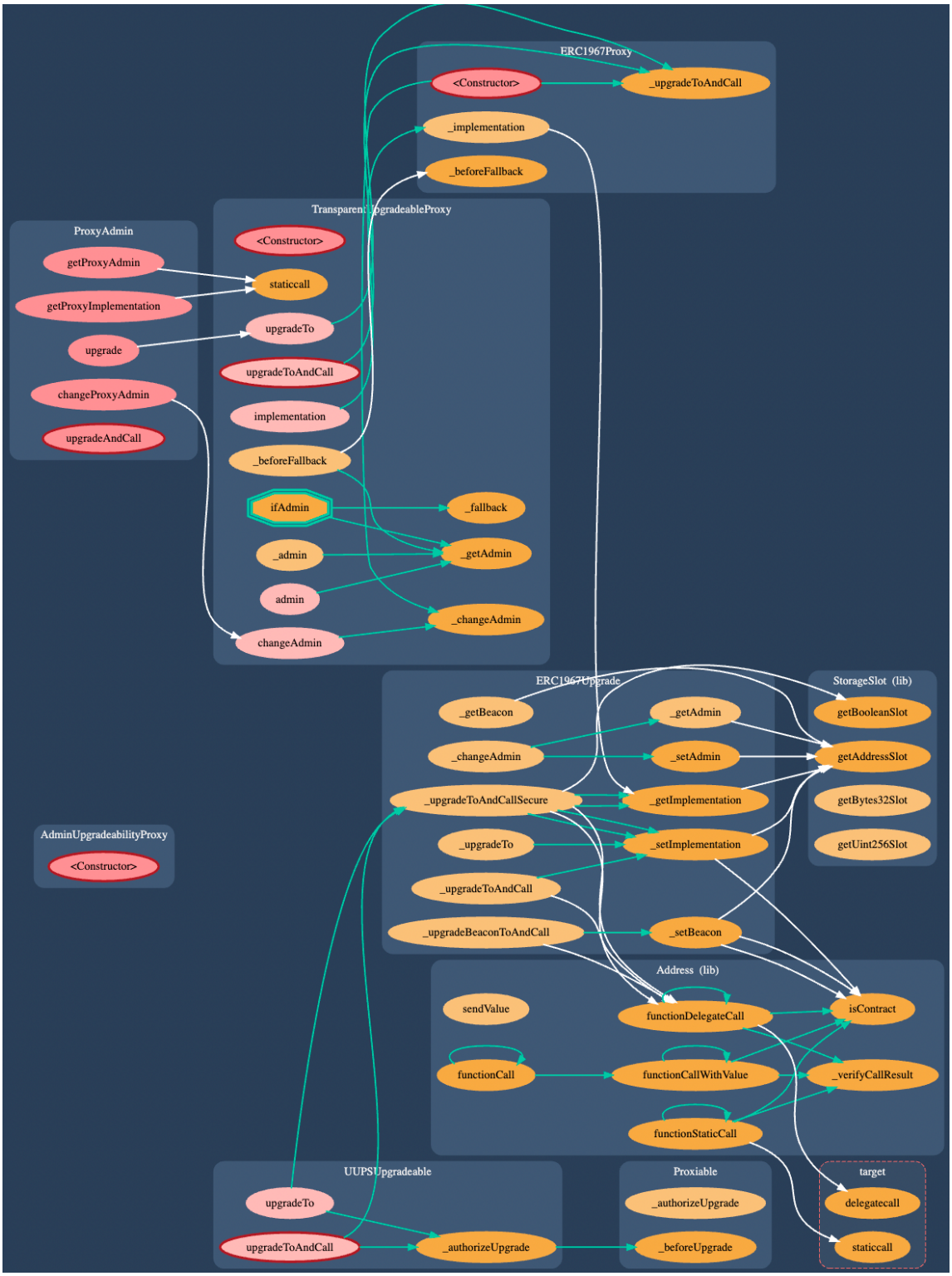
Smart Contract – Dynamic Analysis

Callout Graph

Internal callouts of childofproxiable.sol, context.sol, ownable.sol, IBeacon.sol, and proxy.sol



Internal callouts of ERC1976Proxy, ProxyAdmin.sol, TransparentUpgradeableProxy.sol, ERC1967Upgrade.sol, AdminUpgradabilityProxy.sol, StorageSlot.sol, UUPSUpgradeable.sol, and proxiable.sol



- This contract implements an upgradeable proxy. It is upgradeable because calls are delegated to an * implementation address that can be changed.

```
contract ERC1967Proxy is Proxy, ERC1967Upgrade {
```

- This contract implements a proxy that is upgradeable by an admin.

```
contract TransparentUpgradeableProxy is ERC1967Proxy {
```

- This abstract contract provides a fallback function that delegates all calls to another contract using the EVM instruction `delegatecall`. We refer to the second contract as the _implementation_ behind the proxy, and it has to be specified by overriding the virtual {_implementation} function.

```
abstract contract Proxy {
```

Smart Contract – Manual Analysis

- Active Owner: 0xc0e9f051c6ef76d15f777ae8a7cb14c03550e79e
- When the smart contract has an active owner address, some of the smart contract functions can be edited, modified or altered.
- The KawaFarm smart contracts have an upgradability mechanism. **The smart contract utilizes EIP-1822: Universal Upgradeable Proxy Standard (UUPS)**

```
Abstract contract UUPSUpgradeable is ERC1967Upgrade {
```

The functions included in the smart contract can perform an upgrade of an {ERC1967Proxy}, when this contract is set as the implementation behind such a proxy.

According to KawaFarm, the contract is in beta development. The UUPSUpgradeble mechanism will be used to resolve errors identified during tests, audits, etc.

- KawaFarm smart contract has 1 low severity issue which may or may not create any functional vulnerability.

{

"resource": " /KawaFarm.sol",

"owner": "_generated_diagnostic_collection_name_#0",

"severity": 8, (! Low Severity)

" Expected identifier, got 'LParen",

"source": "solc",

}

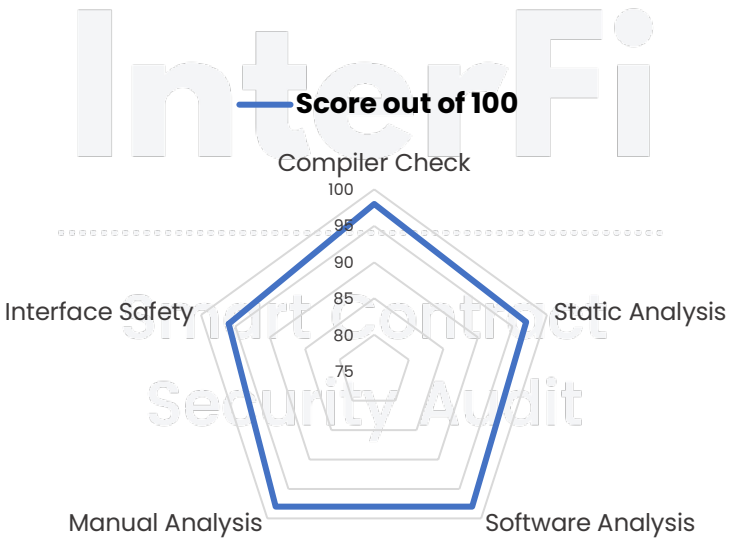
Smart Contract – SWC Attacks

SWC ID	Description	Verdict
SWC-101	Integer Overflow and Underflow	**Passed**
SWC-102	Outdated Compiler Version	**Passed**
SWC-103	Floating Pragma	**Passed**
SWC-104	Unchecked Call Return Value	**Passed**
SWC-105	Unprotected Ether Withdrawal	**Passed**
SWC-106	Unprotected SELFDESTRUCT 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	**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 hardcoded gas amount	**Passed**
SWC-135	Code With No Effects (Irrelevant/Dead Code)	**! Low**
SWC-136	Unencrypted Private Data On-Chain	**Passed**

Smart Contract - Risk Status & Radar Chart

Risk Severity	Status
! Critical	None critical severity issues identified
! High	None high severity issues identified
! Medium	None medium severity issues identified
! Low	**1 low severity issue identified**
Passed	**41 functions and instances verified and passed**



Compiler Check	98
Static Analysis	97
Software Analysis	98
Manual Analysis	98
Interface Safety	96

Auditor's Verdict

InterFi team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analyzed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks.

KawaFarm's smart contract source codes have LOW RISK SEVERITY.

KawaFarm has PASSED the smart contract audit.

KawaFarm is a part of Kawakami Ecosystem. Learn more at: https://kawatoken.io/

InterFi

Smart Contract
Security Audit

Auditor's Note:

- Be aware that active smart contract owner privileges constitute an elevated impact to smart contract's safety and security.
- Project website is not checked due to out of scope. The website hasn't been reviewed for SSL and lighthouse report.

Important Disclaimer

InterFi Network provides contract auditing and project verification services for blockchain projects. The purpose of the audit is to analyse the on-chain smart contract source code, and to provide basic overview of the project. **This report should not be transmitted, disclosed, referred to, or relied upon by any person for any purposes 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 an 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 from external vulnerability, or a hack. Be aware that active smart contract owner privileges constitute an elevated impact to smart contract's 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 on 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 own 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