v

CONTRACT WOLF

Blockchain Security - Smart Contract Audits

Security Assessment

March 28, 2023

Disclaimer

ContractWolf.io audits and reports should not be considered as a form of project's "advertisement" and does not cover any interaction and assessment from "project's contract" to "external contracts" such as Pancakeswap or similar.

ContractWolf does not provide any warranty on its released reports.

ContractWolf should not be used as a decision to invest into an audited project and is not affiliated nor partners to its audited contract projects.

ContractWolf provides transparent report to all its "clients" and to its "clients participants" and will not claim any guarantee of bug-free code within its **SMART CONTRACT**.

ContractWolf presence is to analyze, audit and assess the client's smart contract's code.

Each company or projects should be liable to its security flaws and functionalities.

Scope of Work

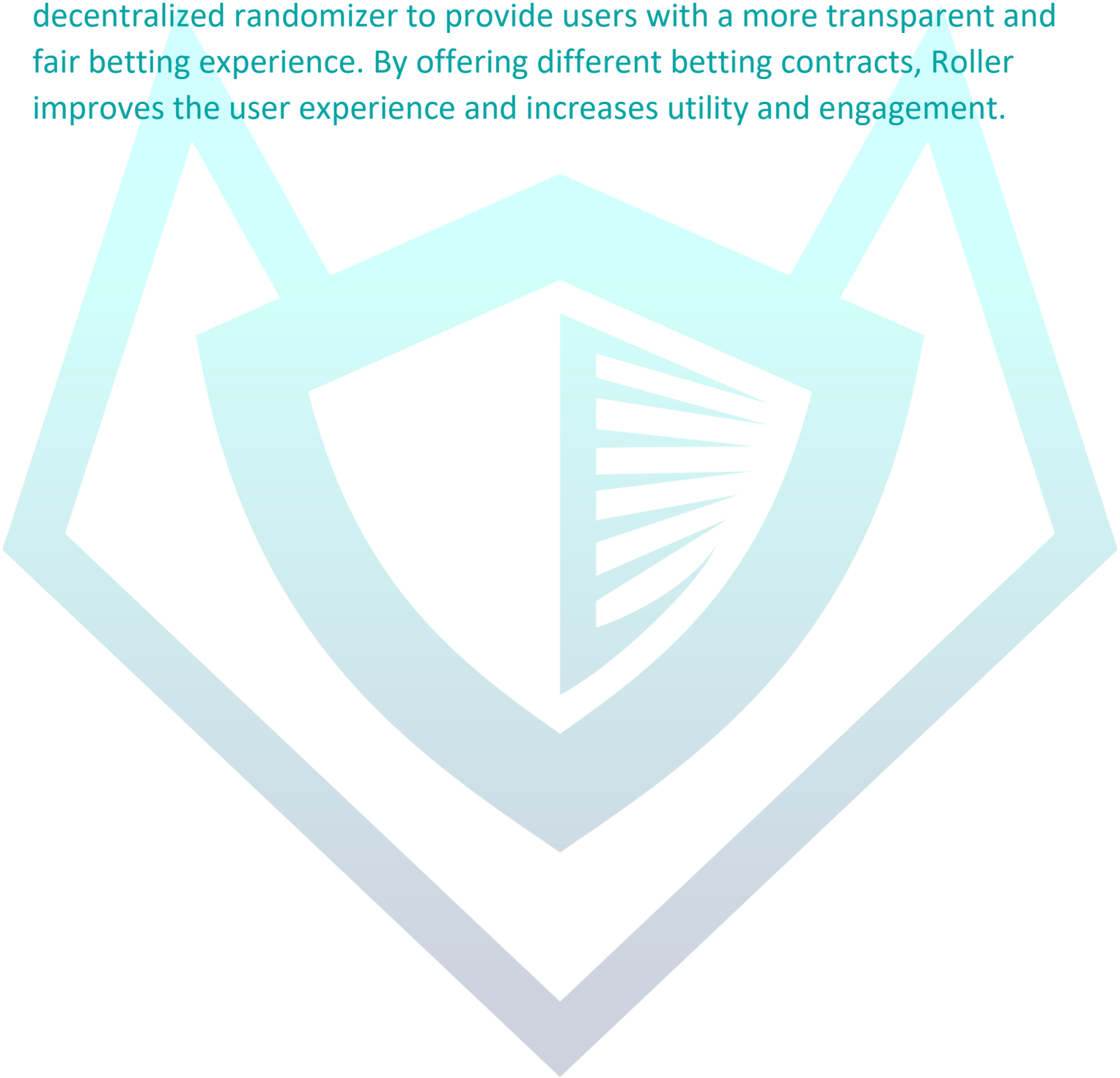
RollerApp team agreed and provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract.

The goal of this engagement was to identify if there is a possibility of security flaws in the implementation of the contract or system.

ContractWolf will be focusing on contract issues and functionalities along with the projects claims from smart contract to their website, whitepaper and repository which has been provided by **RollerApp**.

Description

RollerApp is a P2P betting platform that combines web3 and decentralized randomizer to provide users with a more transparent and fair betting experience. By offering different betting contracts, Roller improves the user experience and increases utility and engagement.



Risk Level Classification

Risk Level represents the classification or the probability that a certain function or threat that can exploit vulnerability and have an impact within the system or contract.

Risk Level is computed based on CVSS Version 3.0

Level	Value	Vulnerability
Critical	9 - 10	An Exposure that can affect the contract functions in several events that can risk and disrupt the contract
High	7 - 8.9	An Exposure that can affect the outcome when using the contract that can serve as an opening in manipulating the contract in an unwanted manner
Medium	4 - 6.9	An opening that could affect the outcome in executing the contract in a specific situation
Low	0.1 - 3.9	An opening but doesn't have an impact on the functionality of the contract
Informational	0	An opening that consists of information's but will not risk or affect the contract

Auditing Approach

Every line of code along with its functionalities will undergo manual review to check its security issues, quality, and contract scope of inheritance. The manual review will be done by our team that will document any issues that there were discovered.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:

    - Review of the specifications, sources, and instructions provided to ContractWolf to make sure we understand the size, scope, and functionality of the smart contract.
    - Manual review of code, our team will have a process of reading the code line-by-line with the intention of identifying potential vulnerabilities and security flaws.

2. Testing and automated analysis that includes:

    - Testing the smart contract functions with common test cases and scenarios, to ensure that it returns the expected results.

3. Best practices review, the team will review the contract with the aim to improve efficiency, effectiveness, clarifications, maintainability, security, and control within the smart contract.

4. Recommendations to help the project take steps to secure the smart contract.

Used Code from other Frameworks/Smart Contracts (Direct Imports)

Imported Packages

- IERC20PermitUpgradeable
- SignedMathUpgradeable
- ECDSAUpgradeable
- SafeMathUpgradeable
- StringUpgradeable
- CountersUpgradeable
- MathUpgradeable
- SafeCastUpgradeable
- AddressUpgradeable
- Initializable
- SafeMath
- IERC20Upgradeable
- IERC20MetadataUpgradeable
- ContextUpgradeable
- ERC20Upgradeable
- EIP721Upgradeable
- ERC20PermitUpgradeable
- ERC20VotesUpgradeable
- RollerDAOV1

Description

Optimization enabled: Yes

Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	2	9	3	5

Exposed Functions

Version	Public	Private	External	Internal
1.0	22	10	16	172

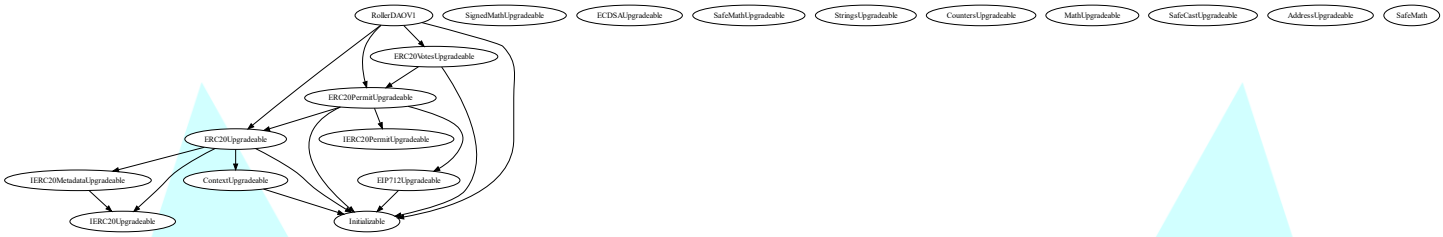
State Variables

Version	Total	Public
1.0	33	0

Capabilities

Version	Solidity Versions Observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	v0.8.0		No	Yes	No

Inheritance Graph



Correct implementation of Token Standard

Tested	Verified
✓	✓

Overall Checkup (Smart Contract Security)

Tested	Verified
✓	✓

Function	Description	Exist	Tested	Verified
TotalSupply	Information about the total coin or token supply	✓	✓	✓
BalanceOf	Details on the account balance from a specified address	✓	✓	✓
Transfer	An action that transfers a specified amount of coin or token to a specified address	✓	✓	✓
TransferFrom	An action that transfers a specified amount of coin or token from a specified address	✓	✓	✓
Approve	Provides permission to withdraw specified number of coin or token from a specified address	✓	✓	✓

Verify Claims

Statement	Exist	Tested	Deployer
Renounce Ownership	–	–	–
Mint	✓	✓	✗
Burn	✓	✓	✗
Block	–	–	–
Pause	–	–	–

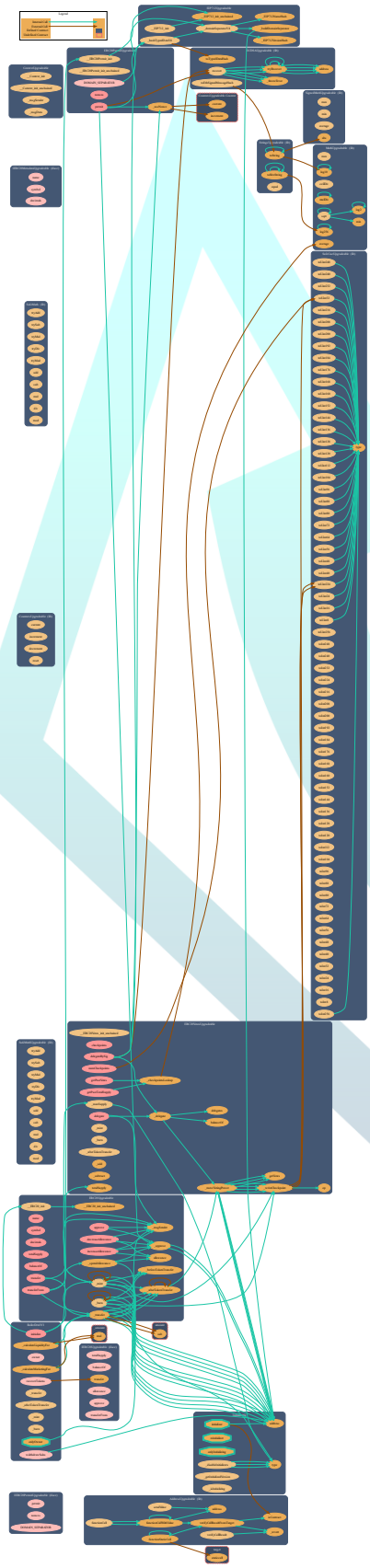
Legend

Attribute	Symbol
Verified / Can	✓
Verified / Cannot	✗
Unverified / Not checked	⚑
Not Available	–

Write Functions of Contract (Testnet)

1. approve (0x095ea7b3)
2. decreaseAllowance (0xa457c2d7)
3. delegate (0x5c19a95c)
4. delegateBySig (0xc3cda520)
5. increaseAllowance (0x39509351)
6. initialize (0xeb990c59)
7. permit (0xd505accf)
8. recoverTokens (0x069c9fae)
9. transfer (0xa9059cbb)
10. transferFrom (0x23b872dd)
11. widthdrawSales (0x04a19d90)

Call Graph



SWC Attacks

ID	Title	Status
SWC-136	Unencrypted Private Data On-Chain	PASSED
SWC-135	Code With No Effects	PASSED
SWC-134	Message call with hardcoded gas amount	PASSED
SWC-133	Hash Collisions with Multiple Variable Length Arguments	PASSED
SWC-132	Unexpected Ether balance	PASSED
SWC-131	Presence of unused variables	PASSED
SWC-130	Right-To Left Override control character (U+202E)	PASSED
SWC-129	Typographical Error	PASSED
SWC-128	DoS With Block Gas Limit	PASSED
SWC-127	Arbitrary Jump with Function Type Variable	PASSED
SWC-126	Insufficient Gas Griefing	PASSED
SWC-125	Incorrect Inheritance Order	PASSED
SWC-124	Write to Arbitrary Storage Location	PASSED
SWC-123	Requirement Violation	LOW ISSUE
SWC-122	Lack of Proper Signature Verification	PASSED
SWC-121	Missing Protection against Signature Replay Attacks	PASSED
SWC-120	Weak Sources of Randomness from Chain Attributes	LOW ISSUE
SWC-119	Shadowing State Variables	PASSED
SWC-118	Incorrect Constructor Name	PASSED
SWC-117	Signature Malleability	PASSED
SWC-116	Block values as a proxy for time	LOW ISSUE
SWC-115	Authorization through tx.origin	PASSED
SWC-114	Transaction Order Dependence	PASSED
SWC-113	DoS with Failed Call	PASSED
SWC-112	Delegate call to Untrusted Callee	PASSED
SWC-111	Use of Deprecated Solidity Functions	PASSED

SWC-110	Assert Violation	**PASSED**
SWC-109	Uninitialized Storage Pointer	**PASSED**
SWC-108	State Variable Default Visibility	**LOW ISSUE**
SWC-107	Reentrancy	**PASSED**
SWC-106	Unprotected SELFDESTRUCT Instruction	**PASSED**
SWC-105	Unprotected Ether Withdrawal	**PASSED**
SWC-104	Unchecked Call Return Value	**PASSED**
SWC-103	Floating Pragma	**LOW ISSUE**
SWC-102	Outdated Compiler Version	**PASSED**
SWC-101	Integer Overflow and Underflow	**PASSED**
SWC-100	Function Default Visibility	**PASSED**

THIS PROJECT WAS AUDITED VIA LOCAL FILE, AND IT IS NOT YET DEPLOYED ON LIVE NET

Low Issues

A floating pragma is set (SWC-103)	L: 3
State variable visibility is not set (SWC-108)	L: 2695, L: 3498, L: 3499, L: 3500, L: 3501, L: 3502
A control flow decision is made based on the block.timestamp environment variable (SWC-116)	L: 3367, L: 3195
Potential use of “block.number” as source of randomness (SWC-120)	L: 3303, L: 3316, L: 3477, L: 3480
Requirement violation (SWC-123)	L: 3516, L: 3610

Findings

Description:

A floating pragma is set (SWC-103)

Suggestion:

Specific version to ensure that the bytecode does not vary between builds.

Description:

State variable visibility is not set (SWC-108)

Suggestion:

Specify variables as **public, internal, or private.**

Description:

A control flow decision is made based on the block.timestamp environment variable (SWC-116)

Suggestion:

Developers should write smart contracts with the notion that block values are not precise, and the use of them can lead to unexpected effects. Alternatively, they may make use oracles.

Description:

Potential use of “block.number” as source of randomness (SWC-120)

Suggestion:

- Using commitment scheme, e.g. RANDAO.
- Using external sources of randomness via oracles, e.g. Oraclize. Note that this approach requires trusting in oracle, thus it may be reasonable to use multiple oracles.
- Using of Bitcoin block hashes, as they are more expensive to mine.

Description:

Requirement Violation (SWC-123)

Suggestion:

If the required logical condition is too strong, it should be weakened to allow all valid external inputs.

Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.

Audit Comments

- Owner cannot burn tokens
- Owner cannot block user
- Owner cannot pause contract
- Owner cannot mint after initial deployment

CONTRACTWOLF

Blockchain Security - Smart Contract Audits