Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

# SMART CONTRACT

## Security Audit Report

Project: Dekasino
Platform: Ethereum
Language: Solidity
Date: March 11th, 2023

# Table of contents




**Email: audit@EtherAuthority.io**

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.


**Email: audit@EtherAuthority.io**

# Introduction

EtherAuthority was contracted by Dekasino Protocol to perform the Security audit of the Dekasino Protocol smart contracts code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on March 11th, 2023.

**The purpose of this audit was to address the following:**

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

# Project Background

- The Dekasino contract is a blockchain based casino game smart contract.
- It is using API3 protocol to acquire a random number from a QRNG provider
- The Dekasino contract inherits the IERC721, IERC20, Ownable, ERC20, IERC20Metadata standard smart contracts from the OpenZeppelin library.
- The Dekasino contract inherits the RrpRequesterV0 standard api from the API3 protocol contracts library.
- These OpenZeppelin and API3 protocol contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

# Audit scope

| **Name** | **Code Review and Security Analysis Report for Dekasino Protocol Smart Contracts** |
|---|---|
| **Platform** | **Ethereum / Solidity** |
| **File 1** | DekasinoRoulette.sol |
| **File 1 MD5 Hash** | D632D05ED56EAE172502C92E4B4379BB |
| **File 2** | BaseVault.sol |
| **File 2 MD5 Hash** | A8177DE608661A9C8E644C259E960B04 |
| **Audit Date** | March 11th,2023 |

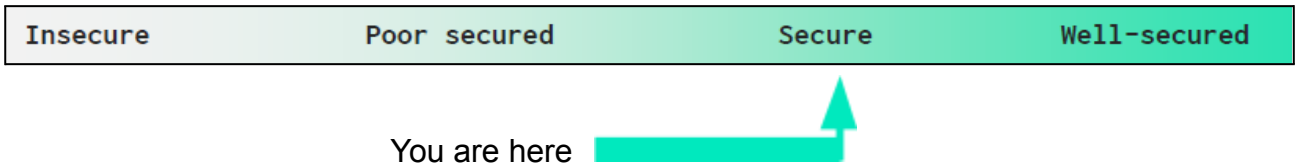

**Email: audit@EtherAuthority.io**

# Claimed Smart Contract Features

| Claimed Feature Detail | Our Observation |
|---|---|
| **File 1 DekasinoRoulette.sol**<br>**Owner Specifications:**<br>• A new token address, vault value, minBet, maxBet, and isSupported status can be set by the owner.<br>• API3 addresses and sponsor wallet addresses can be changed by the owner. | **YES, This is valid.**<br><br>**Owner authorized wallet can set some percentage value and we suggest handling the private key of that wallet securely.** |
| **File 2 BaseVault.sol**<br>• Staking Percent: 3%<br>• Maximum Supply: 0.1 Million<br><br>**Owner Specifications:**<br>• A new vault controller address can be set by the owner.<br>• Staking contract address and stake percentage can be changed by the owner.<br>• A new maximum supply value can be set by the owner. | **YES, This is valid.**<br><br>**Owner authorized wallet can set some percentage value and we suggest handling the private key of that wallet securely.** |

# Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. Also, these contracts do contain owner control, which does not make them fully decentralized.




We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

**We found 0 critical, 1 high, 0 medium and 0 low and some very low level issues.**
**Please be noted that these issues are fixed / acknowledged in the revised contract code.**

**Investors Advice:** Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

# Technical Quick Stats

| Main Category | Subcategory | Result |
|---|---|---|
| Contract Programming | Solidity version not specified | Passed |
| | Solidity version too old | Passed |
| | Integer overflow/underflow | Passed |
| | Function input parameters lack of check | Moderated |
| | Function input parameters check bypass | Passed |
| | Function access control lacks management | Passed |
| | Critical operation lacks event log | Moderated |
| | Human/contract checks bypass | Passed |
| | Random number generation/use vulnerability | N/A |
| | Fallback function misuse | Passed |
| | Race condition | Passed |
| | Logical vulnerability | Passed |
| | Features claimed | Passed |
| | Other programming issues | Passed |
| Code Specification | Function visibility not explicitly declared | Passed |
| | Var. storage location not explicitly declared | Passed |
| | Use keywords/functions to be deprecated | Passed |
| | Unused code | Passed |
| Gas Optimization | “Out of Gas” Issue | Passed |
| | High consumption ‘for/while’ loop | Passed |
| | High consumption ‘storage’ storage | Passed |
| | Assert() misuse | Passed |
| Business Risk | The maximum limit for mintage not set | Passed |
| | “Short Address” Attack | Passed |
| | “Double Spend” Attack | Passed |

**Overall Audit Result:** **PASSED**

## Code Quality

This audit scope has 2 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the Dekasino Protocol are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Dekasino Protocol.

The Dekasino team has not provided unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are **not** well commented on smart contracts.

## Documentation

We were given a Dekasino Protocol smart contract code in the form of a github link. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **not well** commented. But the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Another source of information was its official website: which provided rich information about the project architecture and tokenomics.

## Use of Dependencies

As per our observation, the libraries are used in this smart contracts infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

# AS-IS overview

## DekasinoRoulette.sol

### Functions

| Sl. | Functions | Type | Observation | Conclusion |
|---|---|---|---|---|
| 1 | constructor | read | Passed | No Issue |
| 2 | owner | read | Passed | No Issue |
| 3 | onlyOwner | modifier | Passed | No Issue |
| 4 | renounceOwnership | write | access only Owner | No Issue |
| 5 | _checkOwner | internal | Passed | No Issue |
| 6 | transferOwnership | write | access only Owner | No Issue |
| 7 | _transferOwnership | internal | Passed | No Issue |
| 8 | placeBet | external | Passed | No Issue |
| 9 | _validateBet | internal | Passed | No Issue |
| 10 | setToken | external | Critical operation lacks event log | Acknowledged |
| 11 | setOracle | external | Critical operation lacks event log | Acknowledged |
| 12 | getTotalBetsByUser | external | Passed | No Issue |
| 13 | getTotalBets | external | Passed | No Issue |
| 14 | getBetsOfUser | external | Passed | No Issue |
| 15 | getAllBets | external | Passed | No Issue |

## BaseVault.sol

### Functions

| Sl. | Functions | Type | Observation | Conclusion |
|---|---|---|---|---|
| 1 | constructor | write | Passed | No Issue |
| 2 | owner | read | Passed | No Issue |
| 3 | onlyOwner | modifier | Passed | No Issue |
| 4 | renounceOwnership | write | access only Owner | No Issue |
| 5 | _checkOwner | internal | Passed | No Issue |
| 6 | transferOwnership | write | access only Owner | No Issue |
| 7 | _transferOwnership | internal | Passed | No Issue |
| 8 | name | read | Passed | No Issue |
| 9 | symbol | read | Passed | No Issue |
| 10 | decimals | read | Passed | No Issue |
| 11 | totalSupply | read | Passed | No Issue |
| 12 | balanceOf | read | Passed | No Issue |
| 13 | transfer | write | Passed | No Issue |
| 14 | allowance | read | Passed | No Issue |
| 15 | approve | write | Passed | No Issue |
| 16 | transferFrom | write | Passed | No Issue |
| 17 | increaseAllowance | write | Passed | No Issue |

| | | | | |
|---|---|---|---|---|
| **18** | decreaseAllowance | write | Passed | No Issue |
| **19** | _transfer | internal | Passed | No Issue |
| **20** | _mint | internal | Passed | No Issue |
| **21** | _burn | internal | Passed | No Issue |
| **22** | _approve | internal | Passed | No Issue |
| **23** | _spendAllowance | internal | Passed | No Issue |
| **24** | _beforeTokenTransfer | internal | Passed | No Issue |
| **25** | _afterTokenTransfer | internal | Passed | No Issue |
| **26** | onlyController | modifier | Passed | No Issue |
| **27** | deposit | external | The totalSupply does not increase when it is deposited | Fixed |
| **28** | withdraw | external | Passed | No Issue |
| **29** | lockBet | external | access only Controller | No Issue |
| **30** | unlockBet | external | access only Controller | No Issue |
| **31** | setVaultController | external | Critical operation lacks event log | Acknowledged |
| **32** | setStakingParams | external | Function input parameters lack of check, Critical operation lacks event log | Acknowledged |
| **33** | setMaxSupply | external | Critical operation lacks event log | Acknowledged |
| **34** | getUnderlyingAmount | read | Passed | No Issue |
| **35** | getShareAmount | read | The totalSupply does not increase when it is deposited | Fixed |
| **36** | getUnderlyingBalance | read | Passed | No Issue |
| **37** | getHighWaterMark | read | Passed | No Issue |
| **38** | decimals | read | Passed | No Issue |

# Severity Definitions

| Risk Level | Description |
|---|---|
| Critical | Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc. |
| High | High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial |
| Medium | Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose |
| Low | Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution |
| Lowest / Code Style / Best Practice | Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored. |


**Email: audit@EtherAuthority.io**

# Audit Findings

## Critical Severity

No Critical severity vulnerabilities were found.

## High Severity

(1) The totalSupply does not increase when it is deposited: **BaseVault.sol**

```
function getShareAmount(uint256 _underlyingAmount) public view returns (uint256) {
    return (_underlyingAmount * totalSupply() / getUnderlyingBalance());
}
```

```
function deposit(uint256 _underlyingAmount) external {
    uint256 amountToMint = getShareAmount(_underlyingAmount);
    require(totalSupply() + amountToMint <= maxSupply * scalingFactor, "Max supply exc
    underlying.transferFrom(msg.sender, address(this), _underlyingAmount);
    _mint(msg.sender, amountToMint);
}
```

In the deposit function totalSupply is not increasing, it's always returning 0, So deposit function always minting 0 amount. And totalSupply is not predefined when smart contracts deploy.

**Resolution:** Need to confirm the totalSupply is defined with some specific number of limits, When smart contract deployed.

**Status: This issue is fixed in the revised contract code.**

## Medium

No Medium severity vulnerabilities were found.

## Low

No Low  severity vulnerabilities were found.

## Very Low / Informational / Best practices:

(1) Function input parameters lack of check: **BaseVault.sol**
Function "setStakingParams()" parameter, There is a not set range of the "stakingPercent" variable. This is a percentage calculation variable. That needs to require validation before execution.

**Resolution:** We suggest using validation like percentage type variables, values should have some range like minimum 0 and maximum 100.

**This issue is acknowledged in the revised contract code.**

(2) Critical operation lacks event log:
Missing event log for:
**DekasinoRoulette.sol**

- setToken
- setOracle

**BaseVault.sol**

- setVaultController
- setStakingParams
- setMaxSupply

**Resolution:** Please write an event log for listed events.

**Status: This issue is acknowledged in the revised contract code.**

# Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

### DekasinoRoulette.sol

- setToken: Owner can set a new token address, vault value, minBet, maxBet, isSupported status.
- setOracle: Owner can set a new API3 address, sponsor wallet address.

### BaseVault.sol

- setVaultController: Owner can set a new vault controller address.
- setStakingParams: Owner can set a new staking contract address and new staking percentage.
- setMaxSupply: Owner can set a new maximum supply value.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the airdrop smart contract once its function is completed.

# Conclusion

We were given a contract code in the form of a github link. And we have used all possible tests based on given objects. We had observed 1 high severity issue and some very low severity issues in the smart contracts. These issues are fixed/acknowledged in the revised contract code. **So, the smart contracts are ready for the mainnet deployment**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed contract, based on standard audit procedure scope, is **"Secured"**.


**Email: audit@EtherAuthority.io**

# Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

**Manual Code Review:**

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

**Vulnerability Analysis:**

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

**Documenting Results:**

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

**Suggested Solutions:**

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

# Disclaimers

## EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

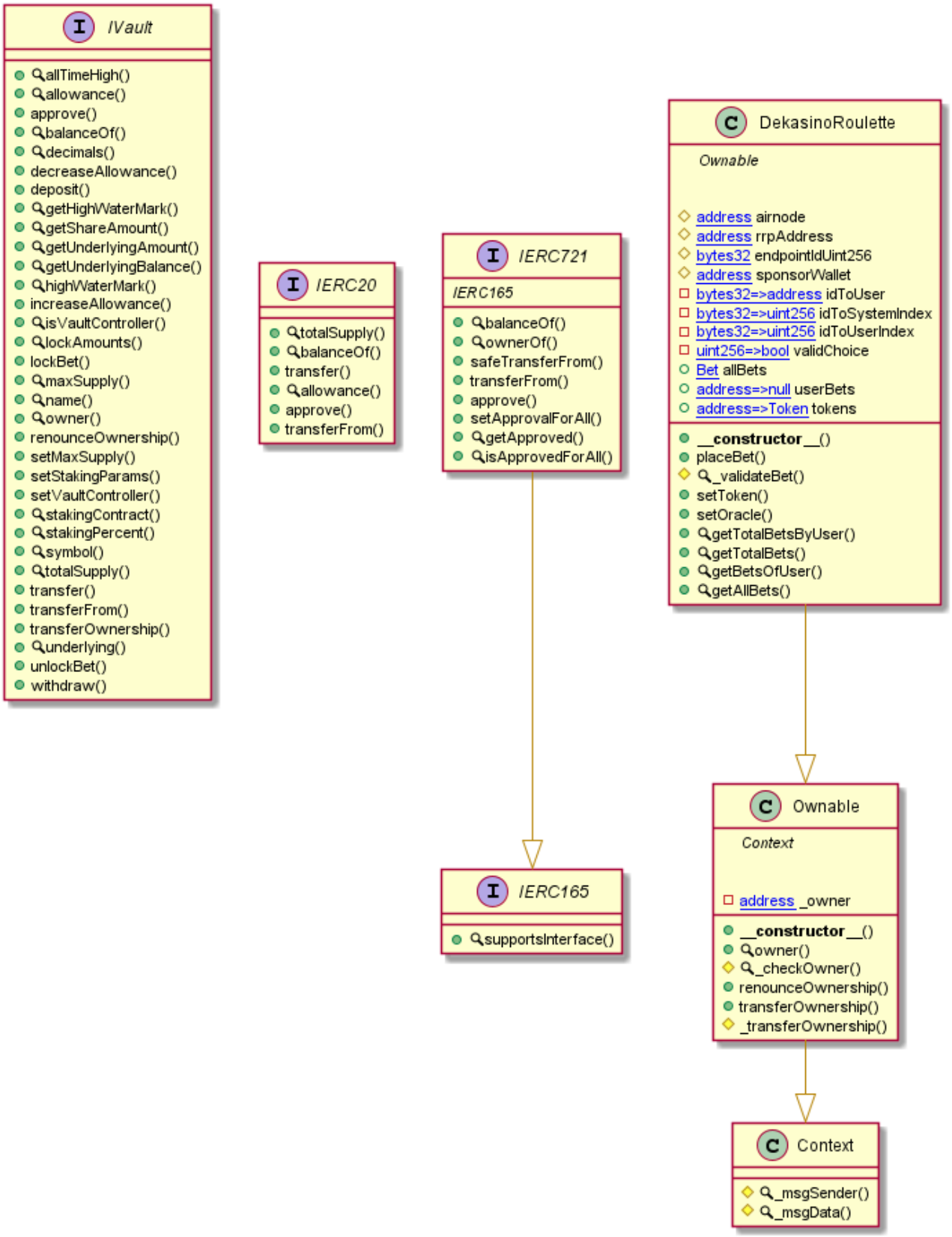
## Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

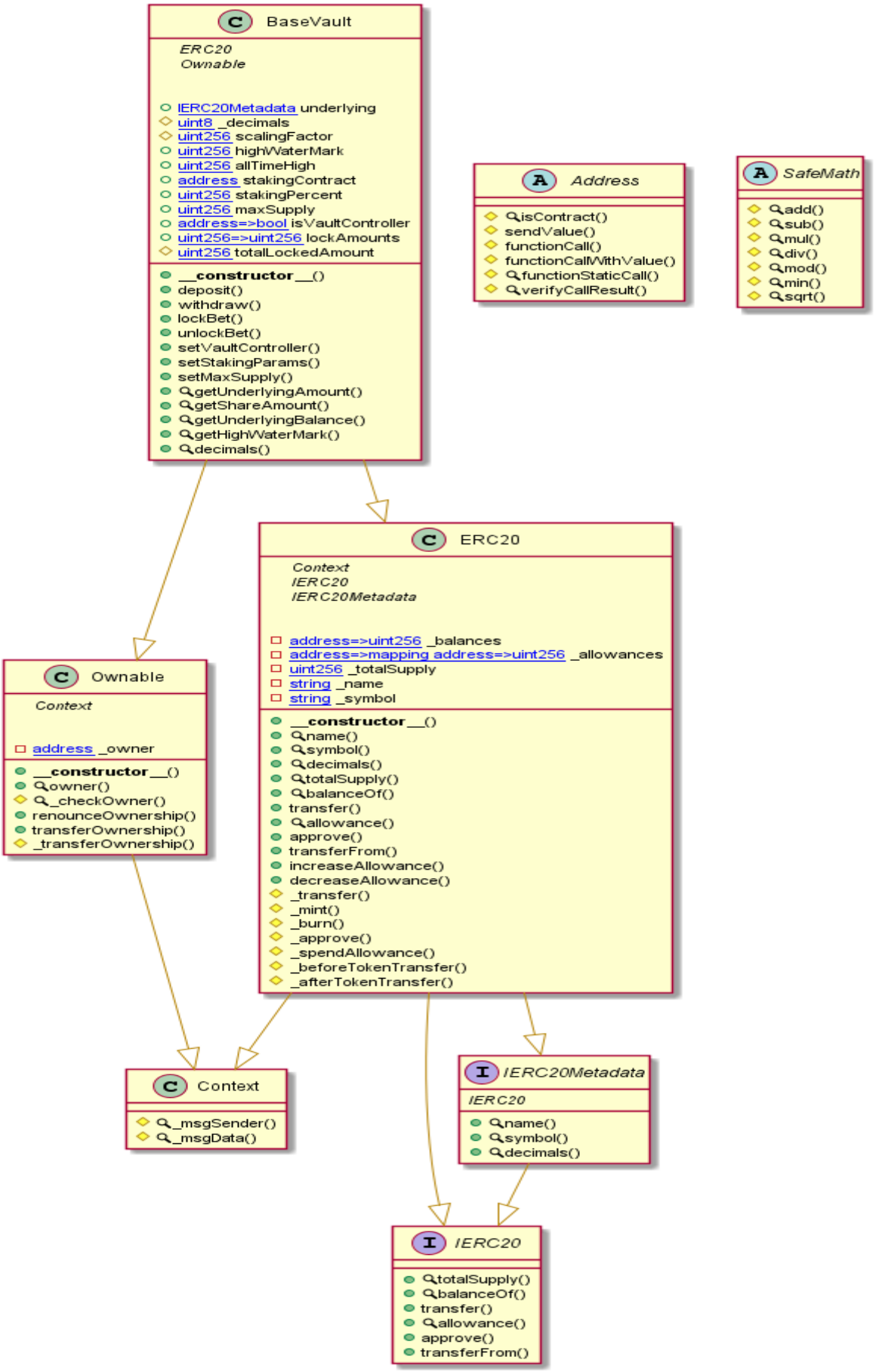
# Appendix

## Code Flow Diagram - Dekasino

## DekasinoRoulette Diagram



# BaseVault Diagram






**Email: audit@EtherAuthority.io**

# Slither Results Log

## Slither log >> DekasinoRoulette.sol

```
IVault.allowance(address,address).owner (DekasinoRoulette.sol#7) shadows:
        - IVault.owner() (DekasinoRoulette.sol#41) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

DekasinoRoulette.setOracle(address,address,bytes32)._airnode (DekasinoRoulette.sol#390) lacks a zero-check on :
                - airnode = _airnode (DekasinoRoulette.sol#391)
DekasinoRoulette.setOracle(address,address,bytes32)._sponsorWallet (DekasinoRoulette.sol#390) lacks a zero-check on :
                - sponsorWallet = _sponsorWallet (DekasinoRoulette.sol#392)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in DekasinoRoulette.placeBet(address,uint8[38]) (DekasinoRoulette.sol#330-341):
        External calls:
        - token.transferFrom(msg.sender,address(this),total) (DekasinoRoulette.sol#334)
        Event emitted after the call(s):
        - BetPlaced(msg.sender,total,_betAmounts,_token,block.timestamp) (DekasinoRoulette.sol#340)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Context._msgData() (DekasinoRoulette.sol#197-200) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.4 (DekasinoRoulette.sol#2) allows old versions
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Parameter DekasinoRoulette.placeBet(address,uint8[38])._token (DekasinoRoulette.sol#330) is not in mixedCase
Parameter DekasinoRoulette.placeBet(address,uint8[38])._betAmounts (DekasinoRoulette.sol#330) is not in mixedCase
Parameter DekasinoRoulette.setToken(address,bool,IVault,uint256,uint256)._token (DekasinoRoulette.sol#371) is not in mixedCase
Parameter DekasinoRoulette.setToken(address,bool,IVault,uint256,uint256)._isSupported (DekasinoRoulette.sol#372) is not in mix
ase
Parameter DekasinoRoulette.setToken(address,bool,IVault,uint256,uint256)._vault (DekasinoRoulette.sol#373) is not in mixedCase
Parameter DekasinoRoulette.setToken(address,bool,IVault,uint256,uint256)._minBet (DekasinoRoulette.sol#374) is not in mixedCas
Parameter DekasinoRoulette.setToken(address,bool,IVault,uint256,uint256)._maxBet (DekasinoRoulette.sol#375) is not in mixedCas
Parameter DekasinoRoulette.setOracle(address,address,bytes32)._airnode (DekasinoRoulette.sol#390) is not in mixedCase
Parameter DekasinoRoulette.setOracle(address,address,bytes32)._sponsorWallet (DekasinoRoulette.sol#390) is not in mixedCase
Parameter DekasinoRoulette.setOracle(address,address,bytes32)._endpointIdUint256 (DekasinoRoulette.sol#390) is not in mixedCas
Parameter DekasinoRoulette.getTotalBetsByUser(address)._user (DekasinoRoulette.sol#396) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (DekasinoRoulette.sol#198)" inContext (DekasinoRoulette.sol#192-201)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

DekasinoRoulette.rrpAddress (DekasinoRoulette.sol#295) is never used in DekasinoRoulette (DekasinoRoulette.sol#268-417)
DekasinoRoulette.idToUser (DekasinoRoulette.sol#299) is never used in DekasinoRoulette (DekasinoRoulette.sol#268-417)
DekasinoRoulette.idToSystemIndex (DekasinoRoulette.sol#300) is never used in DekasinoRoulette (DekasinoRoulette.sol#268-417)
DekasinoRoulette.idToUserIndex (DekasinoRoulette.sol#301) is never used in DekasinoRoulette (DekasinoRoulette.sol#268-417)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

DekasinoRoulette.rrpAddress (DekasinoRoulette.sol#295) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
DekasinoRoulette.sol analyzed (7 contracts with 84 detectors), 28 result(s) found
```

## Slither log >> BaseVault.sol

```
BaseVault.constructor(address,string,string)._name (BaseVault.sol#782) shadows:
        - ERC20._name (BaseVault.sol#435) (state variable)
BaseVault.constructor(address,string,string)._symbol (BaseVault.sol#782) shadows:
        - ERC20._symbol (BaseVault.sol#436) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

BaseVault.setStakingParams(address,uint256) (BaseVault.sol#843-846) should emit an event for:
        - stakingPercent = _newStakingPercent (BaseVault.sol#845)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

BaseVault.setStakingParams(address,uint256)._newStakingContract (BaseVault.sol#843) lacks a zero-check on :
                - stakingContract = _newStakingContract (BaseVault.sol#844)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in BaseVault.deposit(uint256) (BaseVault.sol#791-796):
        External calls:
        - underlying.transferFrom(msg.sender,address(this),_underlyingAmount) (BaseVault.sol#794)
        State variables written after the call(s):
        - _mint(msg.sender,amountToMint) (BaseVault.sol#795)
                - _balances[account] += amount (BaseVault.sol#649)
Reentrancy in BaseVault.unlockBet(uint256,uint256) (BaseVault.sol#814-837):
        External calls:
        - underlying.transfer(msg.sender,_unlockAmount) (BaseVault.sol#821)
        State variables written after the call(s):
        - highWaterMark = getHighWaterMark() (BaseVault.sol#825)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
Pragma version^0.8.4 (BaseVault.sol#2) allows old versions
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (BaseVault.sol#90-95):
        - (success) = recipient.call{value: amount}() (BaseVault.sol#93)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (BaseVault.sol#158-169):
        - (success,returndata) = target.call{value: value}(data) (BaseVault.sol#167)
Low level call in Address.functionStaticCall(address,bytes,string) (BaseVault.sol#187-196):
        - (success,returndata) = target.staticcall(data) (BaseVault.sol#194)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter BaseVault.deposit(uint256)._underlyingAmount (BaseVault.sol#791) is not in mixedCase
Parameter BaseVault.withdraw(uint256)._shareAmount (BaseVault.sol#798) is not in mixedCase
Parameter BaseVault.lockBet(uint256,uint256)._betId (BaseVault.sol#805) is not in mixedCase
Parameter BaseVault.lockBet(uint256,uint256)._lockAmount (BaseVault.sol#805) is not in mixedCase
Parameter BaseVault.unlockBet(uint256,uint256)._betId (BaseVault.sol#814) is not in mixedCase
Parameter BaseVault.unlockBet(uint256,uint256)._unlockAmount (BaseVault.sol#814) is not in mixedCase
Parameter BaseVault.setVaultController(address,bool)._controller (BaseVault.sol#839) is not in mixedCase
Parameter BaseVault.setVaultController(address,bool)._status (BaseVault.sol#839) is not in mixedCase
Parameter BaseVault.setStakingParams(address,uint256)._newStakingContract (BaseVault.sol#843) is not in mixedCase
Parameter BaseVault.setStakingParams(address,uint256)._newStakingPercent (BaseVault.sol#843) is not in mixedCase
Parameter BaseVault.setMaxSupply(uint256)._newSupply (BaseVault.sol#848) is not in mixedCase
Parameter BaseVault.getUnderlyingAmount(uint256)._shareAmount (BaseVault.sol#853) is not in mixedCase
Parameter BaseVault.getShareAmount(uint256)._underlyingAmount (BaseVault.sol#857) is not in mixedCase
Variable BaseVault._decimals (BaseVault.sol#759) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (BaseVault.sol#357)" inContext (BaseVault.sol#351-360)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```

```
BaseVault._decimals (BaseVault.sol#759) should be constant
BaseVault.scalingFactor (BaseVault.sol#760) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

BaseVault.underlying (BaseVault.sol#758) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
BaseVault.sol analyzed (8 contracts with 84 detectors), 61 result(s) found
```

# Solidity Static Analysis

**DekasinoRoulette.sol**

## Security

### Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in DekasinoRoulette.placeBet(address,uint8[38]): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 73:4:

### Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 117:91:

## Gas & Economy

### Gas costs:

Gas requirement of function DekasinoRoulette.getBetsOfUser is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 397:4:

### Gas costs:

Gas requirement of function DekasinoRoulette.getAllBets is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 404:4:



**Email: audit@EtherAuthority.io**

## Miscellaneous

### Constant/View/Pure functions:

DekasinoRoulette._validateBet(address,uint8[38]) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 120:4:

### Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 253:8:

**BaseVault.sol**

## Security

### Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in BaseVault.unlockBet(uint256,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 65:4:

### Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 87:48:

## Gas & Economy

### Gas costs:

Gas requirement of function BaseVault.withdraw is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 49:4:

## Miscellaneous

### Similar variable names:

BaseVault.lockBet(uint256,uint256) : Variables have very similar names "lockAmounts" and "_lockAmount". Note: Modifiers are currently not considered by this static analysis.
Pos: 57:36:

### Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 100:8:

### Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 117:16:

# Solhint Linter

**DekasinoRoulette.sol**

```
DekasinoRoulette.sol:12:19: Error: Parse error: mismatched input '('
expecting {';', '='}
DekasinoRoulette.sol:13:24: Error: Parse error: mismatched input '('
expecting {';', '='}
DekasinoRoulette.sol:14:27: Error: Parse error: mismatched input '('
expecting {';', '='}
DekasinoRoulette.sol:15:20: Error: Parse error: mismatched input '('
expecting {';', '='}
DekasinoRoulette.sol:129:54: Error: Parse error: mismatched input '('
expecting {';', '='}
DekasinoRoulette.sol:133:83: Error: Parse error: mismatched input '('
expecting {';', '='}
DekasinoRoulette.sol:138:22: Error: Parse error: missing ';' at '{'
DekasinoRoulette.sol:142:76: Error: Parse error: mismatched input '('
expecting {';', '='}
DekasinoRoulette.sol:143:57: Error: Parse error: mismatched input '('
expecting {';', '='}
```

**BaseVault.sol**

```
BaseVault.sol:2:1: Error: Compiler version ^0.8.19 does not satisfy
the r semver requirement
BaseVault.sol:33:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
BaseVault.sol:62:45: Error: Avoid to make time-based decisions in
your business logic
BaseVault.sol:87:49: Error: Avoid to make time-based decisions in
your business logic
```

**Software analysis result:**

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.




**Email: audit@EtherAuthority.io**