



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For BetSwirl

10 September 2024



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents	2
Disclaimer	3
1 Overview	4
1.1 Summary	4
1.2 Contracts Assessed	5
1.3 Findings Summary	6
1.3.1 General	7
1.3.2 Bank	7
1.3.3 Game	8
1.3.4 CoinToss	9
1.3.5 Dice	9
1.3.6 Roulette	9
1.3.7 Keno	10
2 Findings	11
2.1 General Issues	11
2.1.1 Issues & Recommendations	12
2.2 Bank	14
2.2.1 Privileged Functions	15
2.2.2 Issues & Recommendations	16
2.3 Game	29
2.3.1 Privileged Functions	29
2.3.2 Issues & Recommendations	30
2.4 CoinToss	50
2.4.1 Issues & Recommendations	51
2.5 Dice	53
2.5.1 Issues & Recommendations	54
2.6 Roulette	55
2.5.1 Issues & Recommendations	56
2.7 Keno	58
2.7.1 Issues & Recommendations	59

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocacy for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for BetSwirl on the Avalanche, Arbitrum One, BNB Chain, Polygon and Base networks. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	BetSwirl
URL	https://www.betswirl.com/
Platform	Avalanche, Arbitrum One, BNB Smart Chain, Polygon and Base
Language	Solidity
Preliminary Contracts	https://github.com/BetSwirl/contracts-v2/commit/482b176ad94e1f5fd9a16c0e8c00a2d6b7247f67
Resolution 1	https://github.com/BetSwirl/contracts-v2/commit/12cdf200e2d9123878cdc99d0910a3ef7ccce591
Resolution 2	https://github.com/BetSwirl/contracts-v2/commit/2bb1c977f7ebf0d1bee4374cf1734db2fdabc557

1.2 Contracts Assessed

Name	Contract	Live Code Match
Bank	0x8FB3110015FBCAA469ee45B64dcd2BdF544B9CFA	✓ MATCH
Game	Dependency	✓ MATCH
CoinToss	0xC3Dff2489F8241729B824e23eD01F986fcDf8ec3	✓ MATCH
Dice	0xAa4D2931a9fE14c3dec8AC3f12923Cbb535C0e5f	✓ MATCH
Roulette	0x6678e3B4AB2a8C8Cdd068F132C21293CcBda33cb	✓ MATCH
Keno	0xc3428E4FEb5C770Db51DCb9B1C08223B10994a89	✓ MATCH

The contract addresses are on the same on all networks — we have only checked that the contracts on the Avalanche network matches the audited code. Users should do the same check on the other networks.



1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● Governance	1	1	-	-
● High	4	4	-	-
● Medium	4	4	-	-
● Low	9	6	1	2
● Informational	34	5	6	23
Total	52	20	7	25

Classification of Issues

Severity	Description
● Governance	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
● High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
● Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
● Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
● Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 General

ID	Severity	Summary	Status
01	HIGH	Reentrancy and DoS risk due to push pattern and native token usage	✓ RESOLVED
02	INFO	Contracts will not work with tokens that rebase or have a fee on transfer	ACKNOWLEDGED

1.3.2 Bank

ID	Severity	Summary	Status
03	MEDIUM	_safeTransfer is vulnerable to reentrancy attacks for native token transfers	✓ RESOLVED
04	MEDIUM	A malicious user might induce losses for the affiliate when withdrawing revenue	✓ RESOLVED
05	LOW	Simultaneous deposits in native and ERC20 tokens should be prevented within deposit()	✓ RESOLVED
06	LOW	Reentrancy inside withdrawProtocolRevenues() allows more funds to be withdrawn than expected	✓ RESOLVED
07	LOW	No maximum threshold in setBalanceRisk()	ACKNOWLEDGED
08	INFO	Incorrect hardcoded values in getTokens()	ACKNOWLEDGED
09	INFO	A minimum allocation rate for the bank should be defined in setHouseEdgeSplit()	ACKNOWLEDGED
10	INFO	Logic for getMaxBetAmount and getMaxBetCount allow users to craft bets which can drain more than the entire balance of Bank in a single bet	✓ RESOLVED
11	INFO	Use AccessControlDefaultAdminRules as a safer alternative to AccessControl from OpenZeppelin	ACKNOWLEDGED
12	INFO	EnumerableSet should be used instead of plain arrays for addToken()	✓ RESOLVED
13	INFO	Typographical issues	✓ RESOLVED
14	INFO	Insufficient validation	PARTIAL
15	INFO	Add more parameters to events emitted in setter functions	PARTIAL
16	INFO	Start and end indexes should be added to iterate through the array in withdrawDividends()	ACKNOWLEDGED
17	INFO	Gas optimizations	PARTIAL

1.3.3 Game

ID	Severity	Summary	Status
18	GOV	Governance privileges	✓ RESOLVED
19	HIGH	ETH balances of a game can get drained easily	✓ RESOLVED
20	HIGH	Users can brick attempts by the admin to call <code>Bank:withdraw()</code> for native ETH when they are eligible for a refund for their bet, preventing the protocol from collecting earnings	✓ RESOLVED
21	HIGH	Bet recipient can avoid losing bets	✓ RESOLVED
22	MEDIUM	Lack of validation of acceptable affiliate house edge amount can allow affiliate to frontrun creation of new bets to jack up fees	✓ RESOLVED
23	LOW	Users can lose out on gains when their bets result in payouts greater than the current balance of the Bank contract	ACKNOWLEDGED
24	LOW	Stuck funds cannot be withdrawn	PARTIAL
25	LOW	<code>refundTime</code> should be configurable	✓ RESOLVED
26	LOW	Bet profit payouts can be send to the 0 address	✓ RESOLVED
27	LOW	<code>withdrawTokenVRFFees</code> should validate if <code>vrfSubId</code> is set	✓ RESOLVED
28	INFO	Affiliates can have a house edge set which is less than the expected minimum value set by admin	ACKNOWLEDGED
29	INFO	Define minimum bet amount within <code>_newBet()</code>	ACKNOWLEDGED
30	INFO	Casting <code>block.timestamp</code> to <code>uint32</code> can break refund logic after the Unix timestamp overflows <code>uint32</code>	ACKNOWLEDGED
31	INFO	<code>refundTime</code> should have a minimum value of 24 hours rather than 12 hours	✓ RESOLVED
32	INFO	Expose <code>_getFees()</code> publicly	ACKNOWLEDGED
33	INFO	Use <code>safeTransfer</code> in <code>_safeNativeTransfer()</code>	ACKNOWLEDGED
34	INFO	Add an absolute maximum for the total bet count per bet	ACKNOWLEDGED
35	INFO	Typographical issues	PARTIAL
36	INFO	Insufficient validation	✓ RESOLVED
37	INFO	Gas optimizations	✓ RESOLVED
38	INFO	Add more parameters to events emitted in setter functions	PARTIAL

1.3.4 CoinToss

ID	Severity	Summary	Status
39	INFO	Implement a function for calculating the maximum bet amount	ACKNOWLEDGED
40	INFO	Gas optimizations	ACKNOWLEDGED
41	INFO	ICoinToss can be placed in a separate file	ACKNOWLEDGED

1.3.5 Dice

ID	Severity	Summary	Status
42	INFO	Implement a function for calculating the maximum bet amount	ACKNOWLEDGED
43	INFO	IDice can be placed in a separate file	ACKNOWLEDGED

1.3.6 Roulette

ID	Severity	Summary	Status
44	INFO	_getPayout() should have a public version	ACKNOWLEDGED
45	INFO	Update getMaxBetCount to better simulate conditions during bet creation	ACKNOWLEDGED
46	INFO	Implement a function for calculating the maximum bet amount	ACKNOWLEDGED
47	INFO	IRoulette can be placed in a separate file	ACKNOWLEDGED

1.3.7 Keno

ID	Severity	Summary	Status
48	MEDIUM	fulfillRandomWords can revert due to highly inefficient logic for picking random numbers	✓ RESOLVED
49	LOW	Bets would be easy to win with high payout factor under some configurations	✓ RESOLVED
50	INFO	Payouts are accumulated even when the number of matching numbers is 0	ACKNOWLEDGED
51	INFO	Implement a function for calculating the maximum bet amount	ACKNOWLEDGED
52	INFO	IKeno can be placed in a separate file	ACKNOWLEDGED





2 Findings



2.1 General Issues

The issues listed in this section apply to the protocol as a whole. Please read through them carefully and take care to apply the fixes across the relevant contracts.



2.1.1 Issues & Recommendations

Issue #01	Reentrancy and DoS risk due to push pattern and native token usage
Severity	 HIGH SEVERITY
Description	<p>Throughout the contracts, there are several issues with the chosen pattern for handling funds. The push pattern means the contract directly sends funds to the owner when a certain action is performed. Furthermore, the native token is a valid token that can be used within the games. As described in several issues later in this report, this means the protocol is vulnerable to reentrancy attacks or DoS attacks that use the return bomb attack.</p> <p>We decided to note this as a general issue as there are many edge-cases where the combination of these two can cause either a reentrancy or DoS situation which is detrimental.</p>
Recommendation	<p>Consider migrating to a pull-pattern where the owner of the assets pulls them out of the contract. Additionally, to minimize the attack surface, we recommend implementing a reentrancy guard on all external functions. We also strongly recommend using only the wrapped native token instead of the native token as a valid asset, with the native asset being used solely to pay for the Chainlink VRF.</p>
Resolution	 RESOLVED <p>The team has stated that push pattern has been kept due to design decision. All re-entrancy and DOS issues have been fixed.</p>

Issue #02	Contracts will not work with tokens that rebase or have a fee on transfer
Severity	 INFORMATIONAL
Description	Across the contracts, tokens that rebase or have a fee on transfer are not supported as balances are typically updated with the amounts received as parameters rather than the amounts resulting from a transfer.
Recommendation	Ensure that only standard tokens are used.
Resolution	 ACKNOWLEDGED The team commented that they will ensure no rebase or taxable tokens are added in the protocol.



2.2 Bank

Bank is a vital element of the architecture. It holds the funds for the casino, handles the distribution of payouts for winning bets and storing the funds for losing ones. It defines the type of tokens that could be used in games and distributes the accumulated fees. The contract facilitates the execution of the following key actions:

- `deposit`: Allows permissioned roles to deposit additional funds into the bank, to increase the available balances used for payouts.
- `withdraw`: Allows permissioned roles to take out funds from the bank. There should be no pending bets and the accumulated fees to the house cannot be withdrawn.
- `addToken`: Allows permissioned roles to manage what types of tokens are accepted for bets in games.
- `setHouseEdgeSplit`: Allows permissioned roles to define the allocation of fees between each party.
- `Withdraw fees`: The contract implements a number of permissioned functions for withdrawing the accumulated fees for different allocations: dividends, bank, affiliate revenue, protocol revenue.
- `payout`: Permissioned function callable only by games when a payout should be sent to a bet recipient.
- `cashIn`: Permissioned function callable only by games to register when amounts of losing bets are sent to the bank.

2.2.1 Privileged Functions

- `deposit[DEFAULT_ADMIN_ROLE or BANKROLL_PROVIDER]`
- `withdraw[DEFAULT_ADMIN_ROLE or BANKROLL_PROVIDER]`
- `setBalanceRisk[DEFAULT_ADMIN_ROLE or BANKROLL_PROVIDER]`
- `addToken[DEFAULT_ADMIN_ROLE]`
- `setAllowedToken[DEFAULT_ADMIN_ROLE]`
- `setPausedToken[DEFAULT_ADMIN_ROLE or BANKROLL_PROVIDER]`
- `setHouseEdgeSplit[DEFAULT_ADMIN_ROLE]`
- `withdrawDividends[DIVIDEND_MANAGER_ROLE]`
- `withdrawDividends[DIVIDEND_MANAGER_ROLE]`
- `payout[GAME_ROLE]`
- `cashIn[GAME_ROLE]`
- `setTeamWallet[DEFAULT_ADMIN_ROLE]`



2.2.2 Issues & Recommendations

Issue #03

_safeTransfer is vulnerable to reentrancy attacks for native token transfers

Severity

 MEDIUM SEVERITY

Description

_safeTransfer() is used to transfer native and ERC20 tokens out of the contract. When it comes to transferring native tokens (such as ETH on mainnet), the following logic is executed:

```
(bool success, ) = user.call{value: amount}("");  
  
if (!success) {  
    // Fallback to wrapped gas token in case of error  
    wrapped.deposit{value: amount}();  
    wrapped.transfer(user, amount);  
}
```

The issue is that a low level call() transfers all of the available gas, giving the recipient enough freedom to execute any logic in its receive() function (including re-entering).

One particular situation where this might prove problematic is with Game._resolvePayout() which calls Bank.payout(), which internally uses _safeTransfer() to send the profit to an untrusted address (the recipient). They can for example consume all the gas and cause Game.fulfillRandomWords() to revert (which should not be allowed happen) or they can re-enter as well.

Recommendation

Since _safeTransfer() falls back to wrapping and sending the wrapped token, it is better to limit the gas in call. This value can be set in a configurable parameter so that the protocol can react to future gas price changes. However, even if it is hardcoded and starts reverting due to insufficient gas for the transfer, it will not be a problem since the wrapping logic will make sure that the recipient always gets the assets.

This is how the fix can look like:

```
uint256 private _callGas
...
function setCallGas (value) {....}
...
(bool success, ) = user.call{value: amount, gas:_callGas}
("");
```

The initial gas can be set to the minimum required for the transfer, which ensures that the receiver cannot re-enter. We also strongly recommend adding a reentrancy guard on every external function.

Resolution



Issue #04**A malicious user might induce losses for the affiliate when withdrawing revenue****Severity** MEDIUM SEVERITY**Description**

`withdrawAffiliateRevenues()` is used to transfer accumulated profits to the affiliate. The issue is that anyone can call this function. There can be multiple cases in which the affiliate might not want to transfer the profits, such as:



- The affiliate is a contract and has a bug in its `ETH receive()` function and needs to be upgraded first before receiving the profits or it will lose them.
- The affiliate is a protocol that got compromised/hacked and until the problem is fixed, all funds sent to it will be lost.

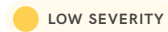
In such cases, the affiliate will not have control over when their own profits get transferred, because anyone can call `withdrawAffiliateRevenues()` at any time.

Recommendation

`withdrawAffiliateRevenues()` should be callable only by the affiliate since it is the recipient of the funds. There is no added value in exposing this function to all users. The same recommendation applies to `withdrawProtocolRevenues()`.

Resolution RESOLVED

Issue #05	Simultaneous deposits in native and ERC20 tokens should be prevented within deposit()
Severity	 LOW SEVERITY
Description	<p>deposit() is used to add assets to the bank. Deposits can be either in native token or ERC20 tokens:</p> <pre> if (_isGasToken(token)) { amount = msg.value; } else { IERC20(token).safeTransferFrom(msg.sender, address(this), amount); } </pre> <p>It is considered good practice in such cases to make sure that only the expected asset gets deposited. The problem here is that on ERC20 transfers, no validation is made if the native token is sent as well.</p>
Recommendation	Consider checking if <code>msg.value > 0</code> on ERC20 transfers and revert — this makes sure the expected asset is deposited and respectively the emitted event indexes the proper amount.
Resolution	 RESOLVED

Issue #06**Reentrancy inside withdrawProtocolRevenues() allows more funds to be withdrawn than expected****Severity****Location**

```
function withdrawProtocolRevenues(address tokenAddress)
external {
    ...
    uint256 treasuryAmount = tokenHouseEdge.treasuryAmount;
    uint256 teamAmount = tokenHouseEdge.teamAmount;
    ...
    _safeTransfer(treasuryWallet, tokenAddress,
treasuryAmount);
    ...
    _safeTransfer(teamWallet, tokenAddress, teamAmount);
}
```

Description

withdrawProtocolRevenues() executes two separate transfers: to treasuryWallet and to teamWallet.

The problem is that the CEI pattern is not followed here and if treasuryWallet is malicious, it can withdraw more native tokens than expected. Here is a scenario:

First call

- treasuryAmount = 10
- teamAmount = 10

The first _safeTransfer() calls receive() on treasuryWallet(), which calls (re-enters) withdrawProtocolRevenues() again.

Second call

- treasuryAmount = 0 - since it was transferred in the first call
- teamAmount = 10 - this is still 10, because the first call has not completed and funds are not yet transferred to teamWallet.

The second call sends 10 ETH to `teamWallet` and ends. This moves the execution flow back to the first call, where the cached value of `teamWallet` is still 10. So 10 ETH are sent again to `teamWallet`. Transaction execution concludes. The deeper the call-stack, the more money will be taken out from the protocol.

This issue is marked as low because the address being called is considered trusted. However, the CEI pattern should still be followed to make sure that the protocol is not exposed to such type of vulnerabilities.

Recommendation Consider following the CEI pattern.

Move this line:

```
uint256 teamAmount = tokenHouseEdge.teamAmount;
```

into the following block:

```
if (teamAmount != 0) {  
    ...  
    _safeTransfer(teamWallet, tokenAddress, teamAmount);  
}
```

Resolution



Issue #07

No maximum threshold in `setBalanceRisk()`

Severity



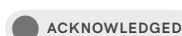
Description

Within `setBalanceRisk()`, there should be a maximum limit on the value of `balanceRisk` to ensure the variable is configured in reasonable ranges.

Recommendation

Since `balanceRisk` depends on the multiplier which is different for each game, a hardcoded value will not work here. A possible approach might be to create a mapping and specify the acceptable maximum values for each game multiplier.

Resolution



Severity

 INFORMATIONAL

Description

The metadata of the native gas token is hardcoded in getTokens():

```
_tokens[i] = TokenMetadata({  
    decimals: 18,  
    tokenAddress: tokenAddress,  
    name: "ETH",  
    symbol: "ETH",  
    token: token  
});
```

As the protocol will be deployed on Polygon, BNB, Arbitrum, Avalanche and BASE, the above configuration would not be correct. For example, the native token on Polygon is MATIC and on Avalanche it is AVAX.



Recommendation

Consider providing the name and symbol parameters dynamically so that they properly reflect the correct information of the native token on each chain.



Resolution



 ACKNOWLEDGED



The client stated: "We are aware of it. But we consider ETH as the native token of each chain. We don't want to change it dynamically depending on the chain. That's how the protocol works."



Issue #09	A minimum allocation rate for the bank should be defined in <code>setHouseEdgeSplit()</code>
Severity	 INFORMATIONAL
Description	<p>When setting different allocations in <code>setHouseEdgeSplit()</code>, the only requirement is that the sum of all allocations is 100% (e.g 10_000).</p> <p>As long as the above condition is met, there are no restrictions on individual allocations being 0. This makes sense for dividend, affiliate, treasury and team since the protocol can decide that no profits should be routed to some of them in favor of other allocations.</p> <p>However, the bank allocation should never be set to 0. This is vital to ensure that the bank is funded with enough assets to pay out to winners. It is recommended there is a minimum value enforced for bank allocation which will ensure protocol integrity and player interests.</p>
Recommendation	Consider defining a minimum allocation value for the bank in <code>setHouseEdgeSplit()</code> .
Resolution	 ACKNOWLEDGED




Issue #10	Logic for getMaxBetAmount and getMaxBetCount allow users to craft bets which can drain more than the entire balance of Bank in a single bet
Severity	 INFORMATIONAL
Description	The functions getMaxBetAmount and getMaxBetCount are meant to prevent users from crafting bets which are unreasonably large. However, in practice, they allow the creation of single bets with maximum payouts that greatly exceed the balance of the Bank. If this happens to pay out, all other users with bets made afterwards will not receive any payout.
Recommendation	Since both functions are used together, consider calculating getMaxBetCount so when multiplied with the output of getMaxBetAmount, it cannot lead to a bet with a maximum payout greater than the entirety of the balance of the Bank contract.
Resolution	 RESOLVED The client stated: "We found 0 ways to drain more than the entire balance of the bank with a single bet. Makes no sense to add an additional check as getMaxBetCount & getMaxBetAmount are already checked and are thought to avoid this situation."

Issue #11	Use AccessControlDefaultAdminRules as a safer alternative to AccessControl from OpenZeppelin
Severity	 INFORMATIONAL
Description	OpenZeppelin recommends in their documentation that contracts inheriting AccessControl should use the AccessControlDefaultAdminRules extension which provides additional safety mechanisms that guard the sensitive DEFAULT_ADMIN_ROLE. Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v5.0.2/contracts/access/extensions/AccessControlDefaultAdminRules.sol
Recommendation	Consider using AccessControlDefaultAdminRules.
Resolution	 ACKNOWLEDGED

Issue #12	EnumerableSet should be used instead of plain arrays for addToken()
Severity	 INFORMATIONAL
Description	Using an EnumerableSet from OpenZeppelin to add and check if tokens exist is more structured and efficient when performing multiple array operations.
Recommendation	Consider using EnumerableSet instead of an array in addToken().
Resolution	 RESOLVED

Issue #13	Typographical issues
Severity	 INFORMATIONAL
Description	<p>It is a best practice to use uppercase for constant and immutable variables for better readability. Consider using uppercase for the following immutable variables:</p> <ul style="list-style-type: none"> - treasuryWallet - wrapped <p>—</p> <p>Consider renaming startTokenBankrollProviderTransfer to setTokenBankrollProviderTransfer because the function can also reset the pending bankroll provider to address(0) and stop the started transfer.</p>
Recommendation	Consider fixing the typographical issues.
Resolution	 RESOLVED

Severity

 INFORMATIONAL

Description

Within the constructor, ensure that the value of the wrappedGasToken address is not 0.

Within deposit(), validate that msg.value == amount for native token deposits to make sure the expected amounts are deposited.

Within deposit(), validate that the token is in the _addedTokens mapping.

Within setAllowedToken() and setPausedToken(), validate that the token is in the _addedTokens mapping.

Within payout(), check that affiliate and fees are not 0.



Within getMaxBetAmount(), validate that the multiplier is above balanceRisk (it will not produce 0) and that it is above 10_000 (as marked in the comments).



Recommendation

Consider implementing the validation recommendations.

Resolution

 PARTIALLY RESOLVED

Issue #15 Add more parameters to events emitted in setter functions	
Severity	 INFORMATIONAL
Description	<p>When updating storage variables, it is a good practice to emit both the previous and the updated values. In the current code, only the new values are emitted. This applies to the following functions:</p> <ul style="list-style-type: none"> - setBalanceRisk - setPausedToken - setAllowedToken - setHouseEdgeSplit - setTeamWallet <p>Within acceptTokenBankrollProviderTransfer(), consider adding the old bankroll provider address to the TokenBankrollProviderTransferAccepted event.</p>
Recommendation	Consider adding the pre-update values as a parameter to the emitted events.
Resolution	 PARTIALLY RESOLVED

Issue #16 Start and end indexes should be added to iterate through the array in withdrawDividends()	
Severity	 INFORMATIONAL
Description	<p>withdrawDividends() iterates through every token and calls withdrawDividend() on it. If the number of tokens becomes too large and gas prices are high, the loop might start reverting due to an out of gas error because it will always try to loop through all tokens.</p> <p>Consider adding start and end parameters to the function so that the caller can control how many tokens it will loop through.</p>
Recommendation	Consider adding start and end range to the loop.
Resolution	 ACKNOWLEDGED

Issue #17 Gas optimizations	
Severity	INFORMATIONAL
Description	<p>Within <code>withdraw()</code>, validate that amount is not 0 and return early to save on unnecessary operations.</p> <p>—</p> <p>Within <code>_allocateHouseEdge()</code>, check for each amount that it is not 0 before calculating the fee cut and updating it in storage to prevent unnecessary operations.</p> <p>—</p> <p>Within <code>withdrawAffiliateRevenues()</code>, check and return early if affiliate is 0 to prevent unnecessary calculations.</p>
Recommendation	Consider implementing the gas optimizations.
Resolution	PARTIALLY RESOLVED



2.3 Game

Game implements the core logic required for each game and is inherited by each separate game implementation. Its main responsibility is to manage bet creation and refund, calculating the amounts that have to be paid out (in case of a win) or cashed in (in case of loss) and also managing the Chainlink configuration.



The contract facilitates the execution of the following important flows:

- Bet creation and refunding - Takes care of bet creation and validating that required constraints are met. Also makes sure bets can be refunded in case the VRF response is late or never comes.
- Configuring Chainlink - Configures the parameters for the Chainlink VRF communication.
- Resolve payouts - Calculates the profits and losses for bets and interacts with the Bank contract to transfer payouts or cash in lost bet amounts.

2.3.1 Privileged Functions

- `setHouseEdge[OWNER]`
- `setAffiliateHouseEdge[OWNER]`
- `pause[OWNER]`
- `setVRFSubId[OWNER]`
- `setChainlinkConfig[OWNER]`
- `setVRFCallbackGasBase[OWNER]`

2.3.2 Issues & Recommendations

Issue #18	Governance privileges
Severity	 GOVERNANCE
Description	<p>The contract owner has full control over several variables that can impact the outcome of a transaction:</p> <ul style="list-style-type: none">- set VRF subscription Id & configuration- set VRF callback gas values- withdraw accumulated VRF fees- set VRF coordinator (which calls <code>fulfillRandomWords</code>)
Recommendation	<p>Consider incorporating a Gnosis multi-signature contract as the owner and ensuring that the Gnosis participants are trusted entities</p> <p>This has already been ensured for the Bank protocol, but it should be considered for the Game contract as well.</p>
Resolution	 RESOLVED

Severity



Description

Each game starts by creating a bet through the `_newBet()` function which has the following logic (irrelevant sections have been omitted for brevity):

```
function _newBet(
    address tokenAddress_,
    address receiver,
    uint256 tokenAmount_,
    uint256 multiplier,
    address affiliate,
    IGame.MultiBetData memory multiBetData
){
    ...
    address tokenAddress = tokenAddress_;
    uint256 tokenAmount = tokenAmount_;
    ...
    bool isGasToken = address(0) == tokenAddress;
    ...
    (
        bool isAllowedToken,
        uint256 maxBetAmount,
        uint256 maxBetCount
    ) = bank.getBetRequirements(tokenAddress,
multiplier);

}
...
if (tokenAmount > maxBetAmount) {
    if (isGasToken) {
        Address.sendValue(
            payable(msg.sender),
            // excess gas token sent
            (tokenAmount - maxBetAmount) * betCount
        );
    }
    tokenAmount = maxBetAmount;
}
```

```
...
chargedVRFCost = isGasToken
    ? msg.value - tokenAmount * betCount
    : msg.value;
...
}
```

The essential steps in the above flow are:

1. User provides the tokenAmount as input
2. maxBetAmount value is calculated
3. tokenAmount is compared with maxBetAmount and if tokenAmount > maxBetAmount, tokenAmount is reduced to maxBetAmount.
4. This is the essential part - if the token is a native token (e.g ETH on mainnet), the difference of (tokenAmount - maxBetAmount) is sent back to the caller

And here is an example of how this could be exploited to drain the contract of its ETH balances:

1. Let's assume CoinToss.sol has 100 ETH balance
 2. The calculated maxBetAmount is 10 ETH and chainlinkVRFCost is 2 ETH
 3. Bob is malicious and decides to create a bet
 4. Bob provides the following parameters:
 - tokenAddress_ = ETH (address(0))
 - tokenAmount_ = 100 ETH
 - betCount = 1 (for simplicity)
 - actual msg.value sent is 12 ETH (maxBet + VRFfee)
 5. _newBet() calculates that 90 ETH (100-10) should be refunded back to Bob and tokenAmount is reduced to 10 ETH
 6. Calculated amount left is enough to cover the fee = 2 ETH (msg.value - tokenAmount) and the transaction succeeds
 7. Bob gets 90 ETH + the chance to win on his deposited 10 ETH or even refund it if the VRF response is delayed
-

Recommendation Generally the idea of sending back excess deposit amounts to users is a bad pattern as that often expands the attack surface and opens the door to different exploitable scenarios.

It should be the player's responsibility to provide the correct amount of funds for the bet — they can calculate this in advance by calling a `getMaxBetAmount()` public function before creating the bet. In case the amount is above the allowed maximum, the protocol should just revert with an error and signal to the user to try again with a lower value.

If the team prefers to stick with the current pattern, then an additional check should be included that makes sure that `msg.value > tokenAmount_ * betCount`.

Resolution



Issue #20

Users can brick attempts by the admin to call `Bank:withdraw()` for native ETH when they are eligible for a refund for their bet, preventing the protocol from collecting earnings

Severity**Description**

`Bank:withdraw()` is used by the admin to withdraw excess token amounts from the Bank contract when the token is paused and there are no outstanding bets for that token. This can be seen in the following code snippet:

```
uint256 roleMemberCount = getRoleMemberCount(GAME_ROLE);
for (uint256 i; i < roleMemberCount; i++) {
    if (
        IGameBank(getRoleMember(GAME_ROLE,
i)).hasPendingBets(token)
    ) {
        revert TokenHasPendingBets();
    }
}
```

There is a check that there are no pending bets for all games where that token is being used.

The issue here is that bets to be refunded are also considered pending. Although anyone is able to call `Game:refundBet()` to send a user a refund when the token is native ETH, this causes `bet.receiver` to waste all the gas in the call, potentially bricking attempts to clear the `token.pendingCount`.

Recommendation

Consider using pull payments rather than push payments, where the refund for a user is simply stored in the `refundBet()` call.

Resolution

Description

`fulfillRandomWords()` directly sends all the accumulated profits and non-rolled bet amounts back to the recipient. This is generally considered an anti-pattern especially when it comes to transferring native tokens such as ETH. The main problem stems from the fact that the execution flow is handed over to the recipient, which gives them an amount of control over the transaction. Aside from re-entering, the recipient can also intentionally revert `receive()` or consume all the gas for the transaction.

In BetSwirl, there is a particular exploitable case related to the above-mentioned vector.

1. `fulfillRandomWords()` calls `_resolvePayout()` which calls `_safeNativeTransfer()`
2. `_safeNativeTransfer()` executes a `call()` forwarding all the available gas to the recipient
3. `_safeNativeTransfer()` falls back to wrapping the token in case the low-level `call()` fails which should prevent a revert
4. This however is not true. Since `call()` sends almost all the available gas to the recipient, they can consume all the gas in `receive()` which will not leave enough gas for `fulfillRandomWords()` to finish executing.

With the authority to revert `fulfillRandomWords()` at will, the recipient can prevent losing bets from getting executed:

1. Recipient monitors the mempool and sees a transaction with a random word that would make him lose a bet
2. They front-runs the transaction by activating an infinite while-loop in their `receive()` function
3. `fulfillRandomWords()` executes next and reverts
4. Recipient refunds his bet and tries again

Recommendation The problem with forwarding all the gas in `_safeNativeTransfer()` has already been outlined in the `Game.sol` section.

An additional recommendation here would be to consider switching from the push to the pull pattern. In other words, create a separate function that bet recipients should call manually to withdraw their profits instead of sending the funds directly in `fulfillRandomWords()`. This would ensure an untrusted address cannot influence the resolution logic of the bets.

Resolution



Issue #22**Lack of validation of acceptable affiliate house edge amount can allow affiliate to frontrun creation of new bets to jack up fees****Severity** MEDIUM SEVERITY**Description**

The logic of `_newBet()` which is triggered on all `wager()` calls requires that the caller specifies an affiliate for which the affiliate edge is set for the bet. Ultimately, this affiliate edge is used to specify the fees taken from the bet payout and paid out as fees to the protocol, bank, and to the affiliate. Because of this, the affiliate is incentivised to increase these fees as high as possible.


One way to do this is for the affiliate to set their fees very low, convincing a user to use them. Then, when the affiliate sees the transaction to create a bet in the mempool, the affiliate can call `setAffiliateHouseEdge()` to significantly increase the fees.

Recommendation

There are two options to solve this issues:

- Let the user specify the affiliate house edge that corresponds with the value that the user reads from the storage. Then check that inside the `_newBet()` against the storage variable — if it differs then just revert. We can have another option to let the user specify `type(uint256).max` if they do not care about this check and just want to choose the storage variable.
- Another option is to allow the user to specify a maximum house edge they are willing to pay, which would be passed to `wager()` and `_newBet()`.

Resolution RESOLVED

Issue #23**Users can lose out on gains when their bets result in payouts greater than the current balance of the Bank contract****Severity** LOW SEVERITY**Description**

The logic in `Game::_resolvePayout` will simply clip the user's payout if their payout amount is greater than the current balance of the Bank contract:

```
{  
    uint256 bankroll = bank.getBalance(tokenAddress);  
    if (bankroll < payout - totalBetAmount)  
        payout = bankroll + totalBetAmount;  
}
```

This can mean that users lose out on their earned payout when the Bank does not have enough funds to pay out their bet.

Recommendation



Consider using pull payments rather than push payments to ensure that users retain their earnings, even when the Bank is temporarily undercollateralized.



By using the pull pattern, users can withdraw assets as much as they are available and then subsequently withdraw more as more assets are deposited.



Resolution ACKNOWLEDGED



The team stated: "This case could happen only if multiple bets are placed at the same time, with max bet amount, high bet count and high balance risk and they win. Which means that the probability of this happening is very quite low (especially since few users take such risks).

In most cases, the balance risk will be 2%. So we take the risk it could happen. It will not be hidden for users. We don't want to allow the bank to be undercollateralized."

Issue #24 Stuck funds cannot be withdrawn	
Severity	 LOW SEVERITY
Description	<code>_resolvePayout()</code> and <code>refundBet()</code> only handle the deposited bet amounts accounted for in <code>_newBet()</code> . However, the contract also defines a <code>receive()</code> function which allows native tokens to get sent accidentally and get stuck. The same applies for ERC20 token donations. There are no functions that allows those funds to be taken out from the contract.
Recommendation	Consider adding an owner-restricted function that calculates the stuck/donated amounts in the contract (by subtracting the deposited bet amounts) and allows them to be withdrawn. Another option is to just remove this function and implement a custom payable function that is used to receive gas tokens.
Resolution	 PARTIALLY RESOLVED

Issue #25 refundTime should be configurable	
Severity	 LOW SEVERITY
Description	The requirements for the duration might change or the team might need to do some trial and error to define the appropriate <code>refundTime</code> based on each chain, type of game and active users. Defining <code>refundTime</code> as immutable takes away that flexibility from the protocol. It is better to have a setter function that allows the variable to be modified when needed.
Recommendation	Declare <code>refundTime</code> as a normal mutable storage variable and add a permissioned setter function that validates the minimum/maximum thresholds in the constructor.
Resolution	 RESOLVED

Issue #26 Bet profit payouts can be send to the 0 address	
Severity	 LOW SEVERITY
Description	<code>_newBet()</code> does not validate if the receiver parameter is <code>address(0)</code> . This allows bets to be created, where the recipient can be set to <code>address(0)</code> . Such types of bets do not make sense and should not be allowed in the protocol.
Recommendation	Check if recipient is <code>address(0)</code> and revert with an error.
Resolution	 RESOLVED

Issue #27 <code>withdrawTokenVRFFees</code> should validate if <code>vrfSubId</code> is set	
Severity	 LOW SEVERITY
Description	Since <code>withdrawTokenVRFFees()</code> can be called by anyone, it should have a proper validation that the fees will not be transferred in case <code>vrfSubId</code> is set to 0.
Recommendation	Consider reverting if <code>vrfSubId</code> is not set.
Resolution	 RESOLVED



Issue #28**Affiliates can have a house edge set which is less than the expected minimum value set by admin****Severity** INFORMATIONAL**Description**

Affiliates can set their house edge using the `setAffiliateHouseEdge` function which checks whether the house edge they set is less than the allowable minimum set by the admin. This is done in the following code:

```
if (affiliateHouseEdge < defaultHouseEdge) {  
    revert HouseEdgeTooLow();  
}
```

There is an issue with this if the admin then sets a higher minimum default house edge using the `setHouseEdge` function, in that the affiliate's house edge will be unaffected, meaning users can then use this affiliate to receive lower fees than expected by the protocol.

Recommendation

Consider adding logic which returns the maximum of the `tokens[token].houseEdge` and the house edge set for the affiliate to the `getAffiliateHouseEdge` function.

Resolution ACKNOWLEDGED

Severity

 INFORMATIONAL

Description

`_newBet()` only checks if the amount of the bet is not 0.

Additionally, a good practice in such cases is to also enforce minimum value on the bet amounts to prevent users spamming the system with multiple bets that have dust amounts such as 1 wei.

Dust amount bets are not providing any benefit to the protocol and it is better to prevent them from existing. This would also reduce the potential attack surface.

Recommendation

Add a reasonable minimum amount for bets that would prevent bets with too low amounts from getting created.

Resolution

 ACKNOWLEDGED

The client stated: "We had a min bet amount in a previous version, but we removed it to make the protocol more gas efficient and more simple."



Severity

 INFORMATIONAL

Location

```
function refundBet(uint256 id) external {  
    Bet storage bet = bets[id];  
    if (bet.resolved || bet.id == 0) {  
        revert NotPendingBet();  
    } else if (block.timestamp < bet.timestamp + refundTime)  
    {  
        revert NotFulfilled();  
    }  
    ...  
}
```

Description


In the `_newBet` function, the bet creation timestamp is stored as `block.timestamp` cast to `uint32`. The issue is that in the future, `block.timestamp` will no longer fit in `uint32`, meaning this value will overflow. This will then break the protocol logic, allowing an attacker to effectively be able to DOS all attempts to gamble by instantly refunding all attempts to call `wager()` for any of the games as seen in the above code snippet.



When `bet.timestamp` overflows, it will be a small number. This in turn means that `block.timestamp` will always be greater than `bet.timestamp + refundTime` which allows any bet to be refunded as soon as it is made.



Recommendation



Cast `block.timestamp` to a larger datatype such as `uint64` in the `Bet` struct.

Resolution

 ACKNOWLEDGED

Issue #31	refundTime should have a minimum value of 24 hours rather than 12 hours
Severity	 INFORMATIONAL
Description	<p>Based on the Chainlink docs, VRF requests remain pending for 24 hours prior to expiring assuming there were issues with the subscription balance prior to the request being made. This means 24 hours is a more reasonable lower bound for refundTime_.</p> <p>https://docs.chain.link/vrf/v2-5/overview/subscription#minimum-subscription-balance</p>
Recommendation	<p>Alter the validation of refundTime_ in the constructor to the following:</p> <pre>require(refundTime_ >= 86_400 && refundTime_ <= 2_592_000, "refundTime must be between 24 hours & 30 days");</pre>
Resolution	 RESOLVED

Issue #32	Expose _getFees() publicly
Severity	 INFORMATIONAL
Description	<p>Since _getFees() calculates the fees that would be accrued on the amounts won based on the current houseEdge, it makes sense to expose it publicly so that players can calculate the potential fees before starting a game.</p>
Recommendation	Consider exposing the function publicly as getFees().
Resolution	 ACKNOWLEDGED

Issue #33	Use safeTransfer in _safeNativeTransfer()
Severity	 INFORMATIONAL
Description	Use safeTransfer instead of transfer when sending wrapped tokens in _safeNativeTransfer() to be more consistent with the rest of the code.
Recommendation	Consider using safeTransfer instead of transfer.
Resolution	 ACKNOWLEDGED The client stated: "safeTransfer is more gas expensive and only wrapped gas token is sent into _safeNativeTransfer."



Severity

INFORMATIONAL

Description

When `_newBet()` is called, a validation is made that `betCount` is not more than `maxBetCount` which is the product of calling `bank.getBetRequirements(tokenAddress, multiplier)`. Based on the multiplier, the currently configured `balanceRisk` and the bank balances the `maxBetCount` calculated. Since the value of `balanceRisk` and bank balances are dynamic and can produce different values based on the configuration, the protocol can consider adding an absolute value for `maxBetCount` on each game so that there will be a cap on the value in case the formula goes above it.

The main consideration here is that in each game, a loop is executed inside `fulfillRandomWords()` (which should not revert) that iterates based on `betCount`. If the calculated `maxBet` amount is overestimated, the absolute cap would prevent creating a bet that might revert due to an out of gas error. This problem is more prevalent in complex games, such as KENO, where `_roll()` function (executed on each loop) has more internal calls and calculations.

The absolute cap would also allow the protocol to easily lower the `maxBetCount` during times of high gas costs instead of having to modify the formula which would be a more complex operation.

Recommendation

Consider if adding an absolute cap variable for `maxBet` would be beneficial for the protocol.

Resolution

ACKNOWLEDGED

Severity

 INFORMATIONAL

Description

It is a best practice to uppercase constant and immutable variables for easier readability. Use uppercase for the following immutable variables:

- bank
- wrapped
- refundTime

—

There is a typographical error in the comment of the Game contract:

```
/// @notice This should be parent contract of each games.
```

It should be

```
/// @notice This should be the parent contract of each game.
```

—

There is a typographical error at the EXTRA_MULTIBET_GAS immutable variable — callack should be callback

—

Consider creating an isActiveBet modifier to reduce duplicating the following logic used in _resolvePayout() and refundBet():

```
if (bet.resolved || bet.id == 0) {  
    revert NotPendingBet();  
}
```

—



Consider renaming pause() to togglePause() since the function is used for both pausing and unpausing.



Recommendation

Consider fixing the typographical issues.

Resolution

 PARTIALLY RESOLVED

Issue #36 Insufficient validation	
Severity	 INFORMATIONAL
Description	<p>Within the constructor, ensure that the value of the wrappedGasToken address variable is not 0.</p> <p>—</p> <p>Within setVRFSubId(), ensure that subId is not 0.</p> <p>—</p> <p>Within setChainlinkConfig():</p> <ul style="list-style-type: none"> - add a minimum value for requestConfirmations - check that keyHash is not 0 - Check that chainlinkWrapper is not address(0)
Recommendation	Consider implementing the validation recommendations.
Resolution	 RESOLVED

Issue #37 Gas optimizations	
Severity	 INFORMATIONAL
Description	<p>Game : _newBet() has the whenNotPaused modifier while all wager() functions also have the same whenNotPaused modifier. Since _newBet() is only called from wager(), consider removing this modifier from _newBet().</p>
Recommendation	Consider implementing the gas optimizations
Resolution	 RESOLVED

Severity

 INFORMATIONAL

Description

When updating storage variables it is a good practice to emit the previous and the updated values. Currently, only the new values are emitted. This applies to the following functions:

- setVRFSUBId
- setHouseEdge
- setChainlinkConfig
- setVRFCallbackGasBase

Consider adding the old bankroll provider address to the TokenBankrollProviderTransferAccepted event within acceptTokenBankrollProviderTransfer().

Recommendation

Consider adding the pre-update values as a parameter to the emitted events.

Resolution



 PARTIALLY RESOLVED

2.4 CoinToss

CoinToss is played with a two-sided coin. The goal of the game is for users to guess whether the lucky coin face will be Heads or Tails.



2.4.1 Issues & Recommendations

Issue #39	Implement a function for calculating the maximum bet amount
Severity	 INFORMATIONAL
Description	<p>The contract already implements a <code>getMaxBetCount()</code> function. In similar fashion, it is a good idea to also add a function that calculates the <code>maxBetAmount</code> which can be named <code>getMaxBetAmount()</code>.</p> <p>This would provide additional convenience so users can test out their parameters before creating a bet</p>
Recommendation	Consider adding a <code>getMaxBetAmount()</code> function.
Resolution	 ACKNOWLEDGED
	<p>The client stated: "Not been fixed because <code>getMaxBetCount</code> will be deleted, it means we don't need to implement <code>getMaxBetAmount</code> any more."</p>

Issue #40 Gas optimizations	
Severity	INFORMATIONAL
Description	<p>Within <code>wagerWithData()</code>, check if the bet input is empty and revert early to prevent unnecessary operations.</p> <p>—</p> <p>Within <code>fulfillRandomWords()</code>, consider putting the following code check at the start of the function:</p> <pre>if (bet.resolved bet.id == 0) { revert NotPendingBet(); }</pre> <p>Doing so would save a lot of gas. Currently, the function rolls all the rounds and does a bunch of unnecessary calculations until it reaches the <code>_resolvePayout()</code> call where the above check is finally executed.</p>
Recommendation	Consider implementing the gas optimizations.
Resolution	ACKNOWLEDGED



Issue #41 ICoinToss can be placed in a separate file	
Severity	INFORMATIONAL
Description	The ICoinToss interface can be placed in a separate file to improve clarity and readability.
Recommendation	Consider placing the ICoinToss interface in a separate file.
Resolution	ACKNOWLEDGED



2.5 Dice

Dice is played with a hundred-sided dice. The goal of the game is to for the player to guess whether the lucky number will be larger than their chosen number.



2.5.1 Issues & Recommendations



Issue #42	Implement a function for calculating the maximum bet amount
Severity	 INFORMATIONAL
Description	<p>The contract already implements a <code>getMaxBetCount()</code> function. In similar fashion, it is a good idea to also add a function that calculates the <code>maxBetAmount</code> which can be named <code>getMaxBetAmount()</code>.</p> <p>This would provide additional convenience so users can test out their parameters before creating a bet</p>
Recommendation	Consider adding a <code>getMaxBetAmount()</code> function.
Resolution	 ACKNOWLEDGED
	<p>The client stated: "Not been fixed because <code>getMaxBetCount</code> will be deleted, it means we don't need to implement <code>getMaxBetAmount</code> any more."</p>



Issue #43	IDice can be placed in a separate file
Severity	 INFORMATIONAL
Description	<p>The IDice interface can be placed in a separate file to improve clarity and readability.</p>
Recommendation	Consider placing the IDice interface in a separate file.
Resolution	 ACKNOWLEDGED

2.6 Roulette

Roulette is played with players placing a bet on whether the outcome of the numbers are high (19–36) or low (1–18). The winnings are then paid to the player who has placed a successful bet.

2.5.1 Issues & Recommendations

Issue #44	<code>_getPayout()</code> should have a public version
Severity	 INFORMATIONAL
Description	<p><code>_getPayout()</code> contains the formula that calculates how much profit a bet will make based on the deposited amount.</p> <p>It would be useful to also expose it publicly so that users can test their bet amounts before entering the game. This will provide convenience to the players.</p>
Recommendation	Consider adding a public version of the <code>_getPayout()</code> function.
Resolution	 ACKNOWLEDGED

Issue #45	Update <code>getMaxBetCount</code> to better simulate conditions during bet creation
Severity	 INFORMATIONAL
Description	<p><code>getMaxBetCount()</code> takes any number as input to calculate the maximum bet count. This does not align with the actual logic inside <code>wager()</code> which restricts the accepted numbers like this:</p> <pre>if (numbers == 0 numbers >= 2 ** MODULO - 1) { revert NumbersNotInRange(); }</pre>
Recommendation	Consider updating <code>getMaxBetCount()</code> to either return 0 or revert (just like <code>wager()</code>) if the <code>numbers</code> parameter is not valid.
Resolution	 ACKNOWLEDGED <p>The client stated: "Not been fixed because <code>getMaxBetCount</code> will be deleted."</p>

Issue #46	Implement a function for calculating the maximum bet amount
Severity	<div data-bbox="456 165 485 197" style="display: inline-block; width: 10px; height: 10px; background-color: #8e7cc3; border-radius: 50%;"></div> INFORMATIONAL
Description	<p>The contract already implements a <code>getMaxBetCount()</code> function. In similar fashion, it is a good idea to also add a function that calculates the <code>maxBetAmount</code> which can be named <code>getMaxBetAmount()</code>.</p> <p>This would provide additional convenience so users can test out their parameters before creating a bet</p>
Recommendation	Consider adding a <code>getMaxBetAmount()</code> function.
Resolution	<div data-bbox="456 663 485 694" style="display: inline-block; width: 10px; height: 10px; background-color: #8e7cc3; border-radius: 50%;"></div> ACKNOWLEDGED

Issue #47	IRoulette can be placed in a separate file
Severity	<div data-bbox="456 1005 485 1037" style="display: inline-block; width: 10px; height: 10px; background-color: #8e7cc3; border-radius: 50%;"></div> INFORMATIONAL
Description	<p>The <code>IRoulette</code> interface can be placed in a separate file to improve clarity and readability.</p>
Recommendation	Consider placing the <code>IRoulette</code> interface in a separate file.
Resolution	<div data-bbox="456 1290 485 1321" style="display: inline-block; width: 10px; height: 10px; background-color: #8e7cc3; border-radius: 50%;"></div> ACKNOWLEDGED



2.7 Keno

Keno is played by choosing up to a maximum allowed count of numbers. Random numbers are then drawn and based on how many of them match the user chosen numbers a payout multiplier is calculated



2.7.1 Issues & Recommendations

Issue #48

fulfillRandomWords can revert due to highly inefficient logic for picking random numbers

Severity

 MEDIUM SEVERITY

Location

```
while (i < config.maxNumbersPlayed) {
    num = randomWord % config.biggestNumber;
    bitmask = 1 << num;

    // Checking if it's a new number
    if ((result & bitmask) == 0) {
        result |= bitmask;
        unchecked {
            ++i;
        }
    }

    // Get a new randomWord for the next iteration
    randomWord =
    uint256(keccak256(abi.encodePacked(randomWord)));
}
```

Description

getNumbersOutOfRandomWord() is used to draw random numbers based on the VRF response, which then get compared against the chosen numbers from the bet.

The logic of the function is inefficient in terms of gas consumption. It uses a while loop that iterates until the maximum number of allowed numbers to pick are chosen.

The gas intensive part comes from the fact that each drawn number must be unique. For example, if config.maxNumbersPlayed is set to 3, the loop will keep on going until three different numbers are picked. The issue is when config.maxNumbersPlayed and config.biggestNumber are equal to or really close to one another.

So if we assume a maxNumbersPlayed configuration of 10 (the maximum) and biggestNumber is also 10, it should continue looping until 10 is reached if the loop has already drawn 5 different numbers. This means that the number of loops required grows exponentially.

-
- If 5 numbers are already drawn, then the loop has a 50% chance of hitting the next non-duplicate number.
 - At 6 it is 40%.
 - At 8 it is 20%


So at number 9, the loop can spend a really long time iterating until it lands on a non-duplicate number. If the bet contains many rounds, each round will run the above loop, which further multiplies gas consumption. It is not too hard for `fulfillRandomWords()` to run out of gas given these conditions.

Recommendation The first recommendation would be to enforce some type of ratio check between `config.maxNumbersPlayed` and `config.biggestNumber` when they are set inside `updateTokenConfig`.

Additionally, consider modifying the random number generation logic to something that is more straightforward and not so gas intensive if possible.

Resolution



Issue #49**Bets would be easy to win with high payout factor under some configurations****Severity** LOW SEVERITY**Description**

If `config.maxNumbersPlayed` and `config.biggestNumber` are really close to each other, it would be easy for players to win bets at maximum payout factors.

Example:

- `maxNumbersPlayed`: 10
- `biggestNumber`: 11

The random number drawing logic will pick 10 different numbers between 1 and 11 and then compare them against the bet numbers.

This means that players who always pick 10 numbers would have ~90% chance of hitting all of the numbers, which leads to a higher payout factor as well.

Recommendation

Make sure to enforce reasonable ratios between the two configuration variables when they are being set.

Resolution RESOLVED

Description

When rolling a bet to see how many numbers have matched, the payout is calculated using the following two parameters:

- `rolled` - the random numbers from VRF
- `bet . numbers` - the numbers that were chosen for this bet

Based on how many of the bet numbers matched the randomly picked numbers from VRF, `_getPayout()` calculates the profit. Depending on the count of `bet . numbers` and how many of those numbers are a match, a different multiplier (factor) is used to form the final payout.

The factors are configured by the owner via `updateTokenConfig()` and stored in the following mapping:

```
_gainsFactor[config.biggestNumber][config.maxNumbersPlayed][played][matched]
```

The peculiar thing about the factors is that a multiplier is assigned even when there are 0 matched numbers. Common sense leads to the conclusion that if no numbers are matched, the bet should lose the deposited amount and not get any payouts.

We used the Keno contract deployed on Polygon to verify the hypothesis with real world configuration.

Calling `gain()` with the following parameters:

- `token`: `address(0)` - e.g native token
- `played`: 5
- `matchCount`: 0

calculates 19861 as a winning factor.

We put the value into the payout formula that is executed in `_roll()` which is called in `fulfillRandomWords()`:

```

function _getPayout(
    address token,
    uint256 betAmount,
    uint256 played,
    uint256 matchCount
) private view returns (uint256 factor) {
    TokenConfig memory config =
tokenConfigurations[token];
    factor =
        (betAmount *
         _gainsFactor[config.biggestNumber]
[config.maxNumbersPlayed][
            played
            ][matchCount]) /
        FACTORPRECISION;
}

```

Assuming a betAmount of 1 ETH, we get this:

$1e18(\text{betAmount}) * 19_861 / 10_000 \Rightarrow 1.9861e18$

This effectively produces a multiplier of 1.986

Another way to understand this issue is to consider the output of `gain()` for the case in which `biggestNumber` is 40 and `maxNumbersPlayed` is 10. Here we consider there to always be 1 `matchCount`, irrespective of the played amount by the user:

played	gain()
10	5386
9	4671
8	4197
7	3924
6	3847
5	4001
4	4501
3	5678
2	8666
1	20000

Intuitively, it does not make sense for the `gainFactor` to be larger when the user plays 10 than when they play 5, as the probability of matching 1 number with 10 numbers played is higher than with playing 5.

Recommendation Ensure the above behavior is expected and planned by the game architect.

In case it is not, consider setting the factor to 0 for the first index in the array of factors when calling `_calculateFactors()` inside `updateTokenConfig()`.

Resolution

ACKNOWLEDGED

The team commented: "We want to reward the user in proportion to his chances of winning. This means that 0 matchCount is a probability greater than 0% for each configuration, and should therefore be rewarded.

Do not forget it's the odds of drawing only ONE number. Not AT LEAST one number. It means when you play 10 numbers, you also have more chances to draw 2, 3, 4 numbers etc compared when playing with 5 numbers."

Issue #51 **Implement a function for calculating the maximum bet amount**

Severity

INFORMATIONAL

Description



The contract already implements a `getMaxBetCount()` function. In similar fashion, it is a good idea to also add a function that calculates the `maxBetAmount` which can be named `getMaxBetAmount()`.

This would provide additional convenience so users can test out their parameters before creating a bet

Recommendation Consider adding a `getMaxBetAmount()` function.

Resolution

ACKNOWLEDGED

Issue #52	IKeno can be placed in a separate file
Severity	 INFORMATIONAL
Description	The IKeno interface can be placed in a separate file to improve clarity and readability.
Recommendation	Consider placing the IKeno interface in a separate file.
Resolution	 ACKNOWLEDGED





PALADIN
BLOCKCHAIN SECURITY