



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For BetSwirl (PvP, Wheel)

29 October 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 PvPGame	7
1.3.2 PvPGamesStore	7
1.3.3 RussianRoulette	8
1.3.4 Wheel	8
1.3.5 CoinTossBattle	8
2 Findings	9
2.1 PvPGame	9
2.1.1 Privileged Functions	9
2.1.2 Issues & Recommendations	10
2.2 PvPGamesStore	18
2.2.1 Issues & Recommendations	19
2.3 RussianRoulette	20
2.3.1 Issues & Recommendations	21
2.4 Wheel	23
2.4.1 Issues & Recommendations	24
2.5 CoinTossBattle	28
2.5.1 Issues & Recommendations	28



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 Chain, Polygon and Base
Language	Solidity
Preliminary Contracts	https://github.com/BetSwirl/contracts-v2/blob/58495d10adb6bb8479c360f93ce31f7c78a361bc/contracts/pvh/Wheel.sol https://github.com/BetSwirl/contracts-v2/tree/58495d10adb6bb8479c360f93ce31f7c78a361bc/contracts/pvp
Resolution 1	https://github.com/BetSwirl/contracts-v2/tree/37be2349062d90391cd94e0f0d9d3cadfd92646c/contracts
Resolution 2	https://github.com/BetSwirl/contracts-v2/commit/2bb1c977f7ebf0d1bee4374cf1734db2fdabc557

1.2 Contracts Assessed

Name	Contract	Live Code Match
PvPGame	Dependency	✓ MATCH
PvPGamesStore	0xfCDda7A4437c2D69e33592582D3793626710070F	✓ MATCH
RussianRoulette	0xf052CcC269E4C25cDF37232f95b31FE183cf4428	✓ MATCH
Wheel	0xdec2A4f75c5fAE4a09c83975681CE1Dd1dff764b	✓ MATCH
CoinTossBattle	0xA79FD774185b5F70e92dcC8F7437E106D648E17F	✓ 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	0	-	-	-
● High	1	1	-	-
● Medium	4	3	-	1
● Low	6	2	-	4
● Informational	8	5	1	2
Total	19	11	1	7

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 PvPGame

ID	Severity	Summary	Status
01	HIGH	Anyone can join on a non-existing bet, which opens the door to multiple exploits of the system	✓ RESOLVED
02	MEDIUM	Funds can be donated to non-existing bet pots	✓ RESOLVED
03	MEDIUM	Affiliates can front-run the call to _resolveBet to increase affiliate fees beyond what users specify as their acceptable upper limit	✓ RESOLVED
04	MEDIUM	Charging the Chainlink fees when the bet is created vs when the randomness is actually requested can result in underpaying / overpaying	ACKNOWLEDGED
05	LOW	A malicious bet creator can DOS participant refunds	ACKNOWLEDGED
06	LOW	Donated pot funds are distributed even if the bet is not played	ACKNOWLEDGED
07	INFO	Insufficient validation	✓ RESOLVED
08	INFO	Gas optimizations	✓ RESOLVED
09	INFO	Bet refund time is not configurable	ACKNOWLEDGED
10	INFO	Typographical issues	PARTIAL

1.3.2 PvPGamesStore

ID	Severity	Summary	Status
11	INFO	Use Ownable2Step as a safer alternative to Ownable	ACKNOWLEDGED
12	INFO	Typographical issues	✓ RESOLVED

1.3.3 RussianRoulette

ID	Severity	Summary	Status
13	MEDIUM	Bet creator can prevent bet finalization in some scenarios	✓ RESOLVED
14	LOW	Lack of enforcing a reasonable upper bound for startsAt can result in bets that might never be completed	✓ RESOLVED
15	INFO	Gas optimizations	✓ RESOLVED

1.3.4 Wheel

ID	Severity	Summary	Status
16	LOW	0 value weight/multiplier configurations allow creating games that always lose	✓ RESOLVED
17	LOW	Potential read-only re-entrancy	ACKNOWLEDGED
18	LOW	_getCallbackGasLimit amount does not cover the required gas for select edge case games, which can lead to out-of-gas errors in the Chainlink VRF callback	ACKNOWLEDGED
19	INFO	Gas optimizations	✓ RESOLVED

1.3.5 CoinTossBattle

No issues found.

2 Findings

2.1 PvPGame

PvPGame implements the core logic required for each game and it gets inherited by each separate game implementation. Its main responsibility is to manage bet creation, refunds, and cancellations, as well as managing the Chainlink configuration.



The contract allows for the execution of the following important flows:


- Bet creation, refunding & cancelation - Takes care of bet creation and validating the 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.

2.1.1 Privileged Functions

- `setHouseEdge[OWNER]`
- `setAffiliateHouseEdge[OWNER]`
- `togglePause[OWNER]`
- `setVRFSubId[OWNER]`
- `setChainlinkConfig[OWNER]`
- `setVRFCallbackGasBase[OWNER]`
- `setDividendManager[OWNER]`
- `withdrawProtocolRevenues[OWNER]`

2.1.2 Issues & Recommendations

Issue #01	Anyone can join on a non-existing bet, which opens the door to multiple exploits of the system
Severity	 HIGH SEVERITY
Description	<p><code>_joinBet()</code> is used to add participants to a created bet that has not yet been launched. The caller pays the configured <code>betAmount</code> and gets added to the <code>seats</code> array. The issue is that the implemented validation does not properly check if the bet actually exists:</p> <pre>if (bet.resolved bet.vrfRequestId != 0) { revert NotPendingBet(); }</pre> <p>The above check only verifies that the bet has not been resolved or launched. The issue is that non-created bets with their default values would also go through the check (<code>resolved=false</code>, <code>vrfRequestId=0</code>). This creates an opportunity to exploit the system in multiple ways.</p> <p>For example:</p> <ul style="list-style-type: none">- Filling max bet seats and launching the bet without paying fees would spend from the VRF fees paid by honest bet creators, since no fees are collected on <code>_joinBet()</code>- Another variation is filling enough seats so that they are above <code>betMinSeats</code> and then calling <code>launchBet()</code> and executing a VRF call without having paid for it
Recommendation	<p>Create a modifier <code>betExists()</code>, that executes the following check:</p> <pre>if (bet.id == 0) revert BetDoesNotExist();</pre> <p>Then add that modifier to all functions that accept bet id.</p>
Resolution	 RESOLVED

Issue #02**Funds can be donated to non-existing bet pots****Severity** MEDIUM SEVERITY**Description**

Within `fundPot()`, the check that verifies if the bet exists is not enough (as explained in the issue above) which will result in funds being donated to non-existing bets.

```
Bet storage bet = bets[id];
if (bet.resolved || bet.vrfRequestId != 0) {
    revert NotPendingBet();
}
```


Recommendation

Create a modifier `betExists()`, that executes the following check:
`if (bet.id == 0) revert BetDoesNotExist()`

Then add that modifier to all functions that accept bet id.

This would prevent accidental donation of funds to bets that do not exist and losing the assets in the process.

Resolution RESOLVED

Issue #03**Affiliates can front-run the call to `_resolveBet` to increase affiliate fees beyond what users specify as their acceptable upper limit****Severity** MEDIUM SEVERITY**Location**

```
address token = betData.token;
uint16 houseEdge = getAffiliateHouseEdge(affiliate, token);
if (houseEdge == 0) {
    revert ForbiddenToken();
}
if (houseEdge > betData.maxHouseEdge) {
    revert HouseEdgeTooHigh();
}
```

Description

When new bets are initially created, `_newBet()` is called, which checks that the `houseEdge` set by the provided affiliate is not greater than `betData.maxHouseEdge` on line 177. However, this does not actually protect the user, as an affiliate is able to simply increase their `houseEdge` prior to the bet actually being resolved in the `_resolveBet()` call. This is because on line 609, `_getFees()` is called to get the affiliate fees, which in turn triggers a call to `getAffiliateHouseEdge()` that just gathers the current affiliate fee. This means that after a bet is placed, an affiliate can greatly increase their set fee, which will directly reduce the payout.

Recommendation

The affiliate fees that are cached in the `_newBet()` call should be used in `_resolveBet()`, rather than re-calling `getAffiliateHouseEdge()` with `_getFees()`.

Resolution RESOLVED

Issue #04**Charging the Chainlink fees when the bet is created vs when the randomness is actually requested can result in underpaying / overpaying****Severity** MEDIUM SEVERITY**Description**

The fees for the Chainlink VRF request are collected when a user initially creates a bet in `_newBet()` on line 187. This is based on the output of `getChainlinkVRFCost()`, which on line 967 we can see is based on the current `tx.gasprice`.

The issue with the way this is done is that the actual triggering of the VRF request with `_launchBet()` can be way in the future when the gas prices are no longer similar. This can result in the bet creator overpaying for VRF fees, or in the worse case severely underpaying, which will result in the collected funds not being enough to actually trigger the VRF request.

Recommendation

The resolution to this issue is tricky, as the UX is much better when the bet creator is the one who pays the VRF fees.


One method to solve this is to have the user who triggered the VRF request pay for the request, but have a cut of the payout similar to `houseEdgeSplit.initiatorAmount` paid back to them.

Another is to collect additional funds from the bet initiator, at some level beyond what the current gas price is, and refund the amount which is not used when the VRF request is actually triggered.

Resolution ACKNOWLEDGED

Team comment: "We were already aware of this issue. Users will pay more or less than they actually spend. But on average, over the long term, Betswirl will receive the right amount of VRF fees paid. Overpaid funds are tracked via our subgraphs and distributed via jackpot/leaderboard/freebets to the community."

Severity

 LOW SEVERITY

Description

The contract implements the `refundBet()` functionality, which is called in case the randomness request did not arrive in the determined timeframe to refund the deposited funds to bet participants.

The issue here is that only the bet creator and contract owner can call this function and a malicious bet creator can block refunds to honest participants by simply not calling the function. The contract owner would then have to step in and free the funds.


As this issue requires several conditions to be fulfilled and does not lead to permanent freezing of funds, we rated it as low.

Recommendation



Consider allowing any participant to be able to call `refundBet()`.



This might require more code and optimizations in case the seats array is too big. In case this approach is not desired, then we recommend the team constantly monitor for such bets and be ready to react in case a malicious creator emerges.

Resolution

 ACKNOWLEDGED

Team comment: "We already monitor and be ready to react in case a malicious creator emerges. The case where the randomness request did not arrive is extremely rare."

Issue #06 Donated pot funds are distributed even if the bet is not played	
Severity	 LOW SEVERITY
Description	<p>The idea of the <code>fundPot()</code> function is to allow the donation of additional funds to a created bet. The assumption here is that the donating account expects the amounts will be distributed among the participants after the game is played. However an advantageous game creator can cancel bets right after <code>fundPot()</code> is called and take the extra assets for themselves.</p> <p>This should not be allowed and instead the donating account should be able to take back the funds in case the bet was not played, i.e. if it was canceled or refunded.</p>
Recommendation	Consider storing the donated amounts through <code>fundPot()</code> in a separate mapping and only distribute it at bet resolution. In all other cases, allow the depositors to take back their donations.
Resolution	 ACKNOWLEDGED <p>Team comment: "We were already aware of this issue. We put a warning message in our frontend. The user who funds a pot is then aware that the funds could not be shared among the winners if the bet is canceled or refunded."</p>

Issue #07 Insufficient validation	
Severity	 INFORMATIONAL
Description	<p><code>_newBet()</code> prevents the creation of bets that have 0 <code>betAmount</code>.</p> <p>Within <code>_joinBet()</code>, ensure that <code>receiver</code> and <code>seatCount</code> are not 0 values.</p> <p>Within <code>fundPot()</code>, check that <code>amount</code> is not 0.</p>
Recommendation	Consider implementing the above recommendations.
Resolution	 RESOLVED

Severity

 INFORMATIONAL

Description

Within `_newBet()`, place the opponents loop check at the beginning to prevent unnecessary logic execution in case there are invalid opponents arrays.

Within the opponents loop of `_newBet()`, use the already defined opponent instead of `opponents[i]` when updating the `opponentsAllowed` mapping.

Within `_transferNFTs()`, check before the loop if `nftContractsLength > maxNFTs` and revert early instead of executing the check on each loop.

Within `withdrawAffiliateRevenues()`, there is no need for the `affiliate != address(0)` check since `msg.sender` is never 0.

Within `claimNFTs()`, the `nfts` array is loaded into memory from storage on line 452, and then this information is retrieved from storage again on line 458, costing unnecessary load operations. Instead, the `nfts` array should be used to gather this information.

The storage layout for `PvPGame` could be optimized by placing `maxNFTs`, `betId`, and `dividendManager` next to each other in the variable layout.

Recommendation

Consider implementing the above mentioned recommendations.

Resolution

 RESOLVED

Issue #09	Bet refund time is not configurable
-----------	-------------------------------------

Severity	<div><div></div> INFORMATIONAL</div>
Description	The refund time is hardcoded to ONE_DAY and there is no way to update it if needed.
Recommendation	Consider adding a setter function to update the refund time if required.
Resolution	<div><div></div> ACKNOWLEDGED</div> <p>Team comment: "There is not enough space remaining in the contract."</p>

Issue #10	Typographical issues
-----------	----------------------



Severity	<div><div></div> INFORMATIONAL</div>
Description	<p>The <code>_tranfer()</code> function is misspelled and should be corrected to <code>_transfer()</code>.</p> <p>—</p> <p><code>withdrawAffiliateRevenues()</code> effectively duplicates the logic of <code>claim()</code> with the exception of an event, which might be incorrect due to affiliates also being betting participants.</p>
Recommendation	Consider fixing the typographical issues.
Resolution	<div><div></div> PARTIALLY RESOLVED</div>



2.2 PvPGamesStore

PvPGamesStore is used to store and update variables related to token management in games. Most importantly it has the functions for whitelisting tokens which can be used with games, as well as specifying how protocol fees will be split among different addresses.



2.2.1 Issues & Recommendations



Issue #11	Use Ownable2Step as a safer alternative to Ownable
Severity	 INFORMATIONAL
Description	The contract uses OwnableUpgradeable for access control. A safer alternative is to use Ownable2StepUpgradeable — instead of directly transferring ownership to the new owner, the transfer only completes when the new owner accepts the ownership transfer.
Recommendation	<p>Consider using Ownable2StepUpgradeable instead of OwnableUpgradeable. This makes ownership transfers much safer and prevents accidental transfer to invalid addresses.</p> <p>Note: When implementing Ownable2StepUpgradeable, make sure to call <code>__Ownable_init(newOwner)</code> in the initializer because Ownable2StepUpgradeable does not call it in <code>__Ownable2Step_init()</code> since v 5.0 and would leave the contract without an owner.</p>
Resolution	 ACKNOWLEDGED <p>Team comment: "The owner of the contract will be a multi-signature wallet."</p>


Issue #12	Typographical issues
Severity	 INFORMATIONAL
Description	<code>getTokensAddresses()</code> has duplicate functionality to the <code>values()</code> function which is available with the <code>EnumerableSet</code> contract. Consider utilizing this latter function rather than re-writing it.
Recommendation	Consider fixing the typographical issues.
Resolution	 RESOLVED

2.3 RussianRoulette

RussianRoulette is a game which allows a large number of players to join the same game. Then, depending on the death ratio specified by the initiator, some players are "killed" in Russian roulette style. The pot is shared between all the survivors.

2.3.1 Issues & Recommendations

Issue #13	Bet creator can prevent bet finalization in some scenarios
Severity	 MEDIUM SEVERITY
Description	<p>The specific thing with the RussianRoulette game is that the bet execution logic is split into two parts:</p> <ul style="list-style-type: none">- It receives and stores a random number- It requires that an additional function <code>pullTriggers()</code> gets called to finalize the flow <p>The issue is that the bet is considered resolved only after the second part of the flow is executed. This creates an opportunity for the bet creator to prevent finalization when the random number is not in their favor.</p> <p>After <code>fulfillRandomWords()</code> gets called, <code>bet.resolved</code> is still <code>false</code> and if no one calls <code>pullTriggers()</code> on the first day, the disadvantaged creator can call <code>refundBet()</code> to prevent losing their stakes.</p>
Recommendation	<p>When requesting randomness, it is considered best practice to execute all the related logic atomically in <code>fulfillRandomWords()</code>. If the array of participants is kept small, there should be no gas issues.</p> <p>However, if the array is expected to get too big and looping through <code>fulfillRandomWords()</code> is not an option, consider marking the bet as resolved at the first stage, since technically after randomness is received, no one should be able to influence the outcome.</p>
Resolution	 RESOLVED

Issue #14**Lack of enforcing a reasonable upper bound for startsAt can result in bets that might never be completed****Severity** LOW SEVERITY**Description**

Bets for RussianRoulette are allowed to specify a maxSeats value which more or less defines the maximum number of players which can participate in this bet. When this number of players is reached, the bet will automatically begin. However, if this amount is never reached, there is also a startsAt value which can be set, wherein if this time is passed and there are at least the minimum number of required players, then this bet can start.

However, if startsAt is set too high and no other users want to join this bet, then the bet can never start. There is also no way to refund the bet once at least one other player has joined.

Recommendation

Consider enforcing that startsAt is only so much greater than the current timestamp.

Resolution RESOLVED

A maximum limit of 90 days has been added.

Issue #15**Gas optimizations****Severity** INFORMATIONAL**Description**

Deleting the values in the array, as is being done on line 339, is unnecessary because it is impossible to reach these values again.

Recommendation

Consider implementing the above recommendations.

Resolution RESOLVED



2.4 Wheel



Wheel is a generic game that allows you to effectively have multiple games in one, where users define the game configuration for their own games.

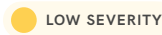
There are many different ways to configure games, both fair and non-fair (fair = weighted average of multipliers = 10000). Each configuration has its own array of multipliers and weights, where the weights denote the probability of getting a given multiplier. The gains for their bet depend on which multiplier was selected based on the randomness received from the Chainlink VRF service. Since there can technically be up to 128 unique multipliers, there is a binary search algorithm which ensures that finding the correct multiplier based on the given weights is efficient.



2.4.1 Issues & Recommendations

Issue #16	0 value weight/multiplier configurations allow creating games that always lose
Severity	 LOW SEVERITY
Description	<p><code>addGameConfig()</code> allows anyone to create different game configurations by combining weight ranges and multipliers. However, the function does not validate for 0 weight and multiplier values, which allows the creation of games that can always lose.</p> <p>Here is an example:</p> <ul style="list-style-type: none">- <code>weight: [10,0,0]</code>- <code>multipliers: [100,1e4+1,0]</code> <p>Based on the above configuration the following gets produced:</p> <ul style="list-style-type: none">- <code>weightRanges: [10,10,10]</code>- <code>totalWeightAmount: 10</code>- <code>totalWeightedMultiplierAmount: 1000</code>- <code>maxMultiplier: 1e4+1 > BP_VALUE</code>- <code>averageWeightedMultiplier: 1000/10 = 100 < BP_VALUE</code> <p>The above parameters will pass all checks and create a valid configuration. As a result, the following <code>_roll()</code> logic will execute:</p> <pre>uint72 rolledWeight = uint72((randomWord % weightRanges[totalColors - 1]));</pre> <p>Since it executes modulo (%) by 10 (the last range element), the result will always be between 0-9 (i.e., below 10), which means that the first multiplier will always get picked. The first multiplier is always 100 or 1% of the <code>betAmount</code>, which means every user that joins that game will always suffer 99% loss.</p>
Recommendation	Validate and revert with an error if some of the elements in the <code>weights_</code> or <code>multipliers_</code> arrays are 0. This way unexpected configurations will be prevented.
Resolution	 RESOLVED <p>Team comment: "Only checking all weights > 0 is enough to fix. Multipliers should be able to be 0."</p>

Issue #17	Potential read-only re-entrancy
Severity	 LOW SEVERITY
Description	<p>Within <code>fulfillRandomWords</code>, once all the bets have been resolved and the respective amounts paid out as a final step, the following storage variable is updated:</p> <pre>wheelBet.rolled = rolledBetsUint8;</pre> <p>This creates a low-severity issue of read-only re-entrancy in situations where the payments happen in native currency. The reason is that the low level <code>_safeNativeTransfer()</code> call is executed before the above storage variable is updated. And if the target contract of the call reads from the <code>wheelBets()</code> view function it will get the variable before the update of the rolled field.</p> <p>The team has already implemented a <code>maxCallGas</code> parameter and the above scenario is only possible if the value of the parameter is high enough to allow logic execution in the target contract</p>
Recommendation	<p>If the above described case is not desirable by team, make sure to properly define <code>maxCallGas</code> so that the target contract cannot execute too much logic or alternatively a <code>nonReentrant</code> guard can be added to the <code>wheelBets()</code> function to make sure it cannot be accessed during bet resolution.</p>
Resolution	 ACKNOWLEDGED <p>Team comment: "The <code>maxCallGas</code> will be quite low. There should be no external logic to call at <code>fulfillRandomWords</code> time. "</p>

Severity**Description**

Based on `wheel.ts`, we assume that the `VRFCallbackGasBase` set for `Wheel` will be `300_000`, and `VRFCallbackGasExtraBet` will be `5_000` at deployment. When gas profiling certain game configurations and other smart contract states, we found that there are certain edge cases in which the call to `fulfillRandomWords` will require more gas than the amount provided by `_getCallbackGasLimit`, meaning the callback will revert for these games. More specifically, we have found this to be the case in the following scenario:

1. User created a config that has `multipliers / weightRanges` arrays with 9 values
2. The random number rolled almost always corresponds to the last / near last of the values in these arrays
3. The user wins overall
4. A high number of bets are placed (here we assume 40, given the `setBalanceRisk`)
5. This is the first bet placed for the `Wheel` contract


The gas required for the callback in this scenario exceeds the amount output by `_getCallbackGasLimit`. Note however that there are other states which also impacts the gas required in the callback as well. For example, all house edge allocations (those set in `setHouseEdgeSplit` being set to non-zero increases the gas required.

We can better understand this scenario based on the fact that the user-winning flow has the most logic, including external calls to the `Bank` contract, therefore requiring more gas than for example losing. Additionally, in this scenario, the `_roll` function has an `if-else` block, which based on the rolled value, can require up to 8 `sload` operations from the `weightRanges` array, which contributes to the required gas.

Ultimately we have found that the callback gas required in this scenario is $> 530_000$, whereas the passed callback gas limit is $300_000 + 40 * 5_000 + 5_500 = 505_500$.

Recommendation Considering the low likelihood of this scenario, it is possible that explicitly handling this scenario is not absolutely required. However, handling this edge case could involve increasing either `VRFcallbackGasExtraBet`, since we have only shown exceeding the gas limit for a large number of bets, or `VRFcallbackGasBase`.

Resolution

 ACKNOWLEDGED

Team comment: "Gas limits in the tests are not especially the ones that will be used in production. But in any case, we will not manage this very rare case to avoid all users paying more VRF fees. Our goal is to do the first bet of the game to test it and configure correctly the VRF callback gas. "

Issue #19 **Gas optimizations**

Severity

 INFORMATIONAL

Description

Inside the loop of `addGameConfig()`, consider using `weight` instead of `weights[i]` for better efficiency and readability.

Consider removing the `nonReentrant` modifier from `setMaxColors()`, there are no external calls in it. This will save additional checks.

Recommendation Consider implementing the above optimizations

Resolution

 RESOLVED

2.5 CoinTossBattle

CoinTossBattle is a game which is played with a two-sided coin. The game's goal is to guess whether the lucky coin face will be Heads or Tails. The payouts are distributed among the winners that joined the game

2.5.1 Issues & Recommendations

No issues found.





PALADIN
BLOCKCHAIN SECURITY