



SMART CONTRACT SECURITY AUDIT

Raffl3

Raffle Contracts

CLIENT

Raffl3

LANGUAGE

Solidity

REVIEW

October 2025

DOCUMENT

Document information

Project	Raffi3, Raffle Contracts
Client	Raffi3
Platform	Not recorded in the original report
Language	Solidity
Repository	LYNC-WORLD/raffle-contracts-evm
Commit	Not recorded in the original report
Methods	Manual code review, static analysis, and best practices evaluation
Review	18 to 28 September 2025
Fix commits	2f68dbf, 27dc251, 9cbc8e5, a47a075, d7bd38a, 6ae74de, 67a53b1

Document history

VERSION	DATE	CHANGE
1.0	October 2025	Original report issued View original report
1.1	October 2026	Reformatted into the Safe Edges standard template. Findings unchanged

This report was reformatted into the Safe Edges standard template in October 2026. Findings, severities, recommendations and code references are unchanged from the original report, which is preserved unchanged in the [Safe Edges Audit Portfolio on GitHub](#).

CONTENTS

Contents

01	Executive summary	4
02	Scope	5
03	Methodology	6
04	Severity and status	7
05	Findings summary	8
06	Detailed findings	10
	C-01 Lack of winner address validation in RafflePool._fulfillTierPool() can potentially locks user's funds in pool contract forever	10
	C-02 Improper access control in RaffleVrfRouter.requestRng can cause pool DoS	11
	C-03 Biased and Flawed Winner Selection Mechanism in RafflePool._fulfillTierPool	12
	H-01 Absence of Robust Signature Replay Protection in SignatureVerifier.verifySignature	13
	H-02 Incomplete Processing Fee Calculation in RafflePool.getTotalPrizeForUser and RafflePool.claim- Prize leads to revenue loss	15
	M-01 Missing validation on prizeAmounts total in PoolTierManager.updateTierConfig	16
	M-02 Pool ownership tracking broken in RaffleFactory.createPool function	17
	L-01 Missing validation on maxTicketsPerAddress relative to totalTickets in RaffleFactory._validatePoolParams	18
	L-02 Missing check whether _owner is wallet address or not in RaffleVrfRouter contract	19
	L-03 Missing validation on totalTickets in RaffleFactory._validatePoolParams	20
	L-04 Duplicate Ticket Numbers in Single Purchase Cause ticketsSold Accounting Discrepancy in Raffle- Pool.buyTickets	20
	L-05 False Event Emission in RafflePool._fulfillTierPool	21
	L-06 Missing endTime Check Before Drawing in RafflePool.requestDraw	22
	L-07 Potential Out-Of-Gas Risk in RafflePool._fulfillTierPool	22
	I-01 Ineffective onlyValidFactory Modifier	23

I-02 Lack of Ticket Availability Function in RafflePool Causes Failed Purchases and Wasted Gas	24
I-03 Incomplete Validation of ticketPrice in RaffleFactory._validatePoolParams	25
I-04 Use Ownable2StepUpgradeable instead of OwnableUpgradeable	25
I-05 Few Important State Change Lacks Event Emission	25
I-06 RaffleVrfRouter.myCallback does not follow the CEI pattern	26
I-07 No Refund Mechanism for Users	27
G-01 Avoid redundant logic and unnecessary state variables to reduce gas	27
G-02 Unused Error	29
G-03 Dead Code	29
Disclaimer	30

Executive summary

About Raffl3

Raffl3 runs on-chain raffles. A factory deploys raffle pools with prize tiers, users buy numbered tickets, and winners are drawn using randomness from a VRF router.

Summary of the review

Safe Edges reviewed the Raffl3 raffle contracts (RaffleFactory, RafflePool, PoolTierManager, SignatureVerifier and RaffleVrfRouter) between 18 and 28 September 2025. Twenty-four issues were identified: three Critical, two High, two Medium, seven Low and ten Informational, three of which are gas optimisations. The Critical issues concerned unvalidated winner addresses that could lock user funds, missing access control on VRF requests that could block pools, and biased winner selection. All Critical, High, Medium and Low issues were fixed; of the Informational issues, five were fixed and five acknowledged.

Findings by severity



Findings by status

STATUS	ISSUES
Resolved	19
Acknowledged	5

Scope

Repository	LYNC-WORLD/raffle-contracts-evm
Commit	Not recorded in the original report
Platform	Not recorded in the original report
Language	Solidity

IN-SCOPE FILES

```
RaffleFactory  
RafflePool  
PoolTierManager  
SignatureVerifier  
RaffleVrfRouter
```

Only the code at the commit above was reviewed. Changes made after this commit are not covered by this report.

Methodology

Methods recorded for this engagement: Manual code review, static analysis, and best practices evaluation.

Manual review

Auditors read the in-scope code line by line, tracing control flow, state changes and trust boundaries against the intended behaviour described by the client.

Threat modelling

We map privileged roles, external dependencies and value flows, then test the assumptions the code makes about each of them.

Functional and automated testing

Where tooling exists for the language and chain, we run static analysis and targeted tests to confirm suspected issues and check edge cases.

Fix review

After the client addresses the findings, we review the updated code and record the final status of each issue.

Severity and status

Severity levels

SEVERITY	DEFINITION
Critical	Direct loss or permanent lock of funds, or full compromise of the protocol, with few or no preconditions. Must be fixed before deployment.
High	Significant loss of funds, broken core functionality or manipulation of critical state under realistic conditions. Should be fixed before deployment.
Medium	Limited loss or degraded functionality, or issues that need specific conditions or privileged access to exploit. Should be fixed.
Low	Minor impact, low likelihood, or deviations from best practice. Fix when practical.
Informational	Code quality, documentation, gas and style observations with no direct security impact.

Status definitions

STATUS	DEFINITION
Resolved	The issue was fixed and the fix was verified.
Partially resolved	The risk was reduced but not fully removed.
Acknowledged	The client accepted the issue without changing the code.
Open	The issue was not addressed at the time of the report.
Not recorded	The original report did not record a resolution status for this issue.

Findings summary

ID	TITLE	SEVERITY	STATUS
C-01	Lack of winner address validation in RafflePool._fulfillTierPool() can potentially locks user's funds in pool contract forever	Critical	● Resolved
C-02	Improper access control in RaffleVrfRouter.requestRng can cause pool DoS	Critical	● Resolved
C-03	Biased and Flawed Winner Selection Mechanism in RafflePool._fulfillTierPool	Critical	● Resolved
H-01	Absence of Robust Signature Replay Protection in SignatureVerifier.verifySignature	High	● Resolved
H-02	Incomplete Processing Fee Calculation in RafflePool.getTotalPrizeForUser and RafflePool.claim- Prize leads to revenue loss	High	● Resolved
M-01	Missing validation on prizeAmounts total in PoolTierManager.updateTierConfig	Medium	● Resolved
M-02	Pool ownership tracking broken in RaffleFactory.createPool function	Medium	● Resolved
L-01	Missing validation on maxTicketsPerAddress relative to totalTickets in RaffleFactory._validatePoolParams	Low	● Resolved
L-02	Missing check whether _owner is wallet address or not in RaffleVrfRouter contract	Low	● Resolved
L-03	Missing validation on totalTickets in RaffleFactory._validatePoolParams	Low	● Resolved
L-04	Duplicate Ticket Numbers in Single Purchase Cause ticketsSold Accounting Discrepancy in Raffle-Pool.buyTickets	Low	● Resolved
L-05	False Event Emission in RafflePool._fulfillTierPool	Low	● Resolved
L-06	Missing endTime Check Before Drawing in RafflePool.requestDraw	Low	● Resolved

L-07	Potential Out-Of-Gas Risk in RafflePool._fulfillTierPool	Low	● Resolved
I-01	Ineffective onlyValidFactory Modifier	Informational	● Resolved
I-02	Lack of Ticket Availability Function in RafflePool Causes Failed Purchases and Wasted Gas	Informational	● Acknowledged
I-03	Incomplete Validation of ticketPrice in RaffleFactory._validatePoolParams	Informational	● Acknowledged
I-04	Use Ownable2StepUpgradeable instead of OwnableUpgradeable	Informational	● Acknowledged
I-05	Few Important State Change Lacks Event Emission	Informational	● Acknowledged
I-06	RaffleVrfRouter.myCallback does not follow the CEI pattern	Informational	● Resolved
I-07	No Refund Mechanism for Users	Informational	● Acknowledged
G-01	Avoid redundant logic and unnecessary state variables to reduce gas	Informational	● Resolved
G-02	Unused Error	Informational	● Resolved
G-03	Dead Code	Informational	● Resolved

Detailed findings

C-01 Lack of winner address validation in RafflePool._fulfillTierPool() can potentially locks user's funds in pool contract forever

Severity **Critical** Status ● Resolved Fix commit 2f68dbf

DESCRIPTION

In `_fulfillTierPool()`, winning ticket number is calculated as :

```
uint256 randomIndex = rng[rngIndex] % ticketsSold;
uint256 selectedTicket = randomIndex;
// existing code
address winner = ticketOwner[selectedTicket];
```

The The function assumes that `ticketOwner[selectedTicket]` are valid purchased tickets.

However, in `buyTickets`, users can select arbitrary ticket numbers within the `totalTickets` range:

```
if (ticketNumber >= tierParams.totalTickets) {
    revert TicketNumberOutOfRange();
}
```

This means the `selectedTicket` may correspond to a ticket that was never actually sold, resulting in `ticketOwner[selectedTicket]` being `address(0)`. Under the current logic, such an entry can be incorrectly assigned as a winner. The severity of this issue is compounded by the fact that these prize funds become permanently inaccessible:

the only recovery mechanism, `emergencyWithdraw()`, is restricted to pools in the `Cancelled` state. However, the pool is already into `Settled` state before processing winners (`_fulfillTierPool()`), and settled pools cannot be cancelled.

As a result, even the owner is unable to withdraw the locked funds, leaving them stuck in the contract.

```
function cancelPool() external onlyOwner {
    @> if (state == ILuckyBidsTypes.PoolState.Settled)
        revert PoolNotLive();
    ....}

function emergencyWithdraw() external onlyOwner {
    @> if (state != ILuckyBidsTypes.PoolState.Cancelled)
        revert PoolNotLive();
    ....}
```

IMPACT

This issue would leads to unclaimed prizes, unfair prize distribution, stuck funds in the pool

C-02 Improper access control in RaffleVrfRouter.requestRng can cause pool DoS

Severity **Critical** Status ● Resolved Fix commit 27dc251

DESCRIPTION

The requestRng function checks whether the caller matches the pool address provided in the calldata. However, it does not verify that the pool is a legitimate contract created or registered by RaffleFactory. As a result, a malicious contract could call requestRng and bypass this validation, successfully triggering the function. While this may appear harmless since the function only requests random numbers, each successful call increments the global requestId, which can lead to unintended consequences for legitimate pools.

```
function requestRng(
    address pool,
    uint8 rngCount
) external returns (uint256) {
    if (!active) revert computeBorrows();
    @>> if (msg.sender != pool) revert UnauthorizedCaller();
    @>> uint256 requestId = ++requestCounter;
}
```

This requestId is then passed to the pool in the callback via:

```
function myCallback(uint256 nonce, uint256[] calldata
    rngList) external {
    if (msg.sender != supraRouter) revert
    OnlySupraRouterCanCall();
    address pool = nonceToPool[nonce];
    if (pool == address(0)) revert InvalidRequest();
    uint256 requestId = nonceToRequestId[nonce];
    (bool success, ) = pool.call(
        abi.encodeWithSignature(
    @>>         "fulfillRandomness(uint256,uint256[])",
    @>>         requestId,
            rngList
        )
    );
}
```

Because requestId is global and not scoped per pool, any external caller can increment the global counter. This can cause the requestId received by a legitimate pool to exceed the pools currentRngRequestId sequential check.

```
@>> if (requestId == 0 || requestId > currentRngRequestId) {
    revert InvalidRequestId();
}
```

As a result, legitimate randomness requests can fail, leading to a denial-of-service for the pool.

IMPACT

Attacker can increment the routers requestId counter to block legitimate pools from receiving randomness. Legitimate pool cannot complete raffle operations

RECOMMENDED MITIGATION

Consider checking whether pool is legitimate or not.

```
+ import {IRaffleFactory} from "../interfaces/IRaffleFactory.sol";
+ address public factory;
+ error InvalidPool();
function initialize(
    address _supraRouter,
    address _owner,
+   address _factory
) public initializer {
    __Ownable_init(_owner);
    __UUPSUpgradeable_init();
+   factory = _factory;
    supraRouter = _supraRouter;
    active = true;
}
function requestRng(
    address pool,
    uint8 rngCount
) external returns (uint256) {
    if (!active) revert computeBorrows();
    if (msg.sender != pool) revert UnauthorizedCaller();
+   if (!IRaffleFactory(factory).isValidPool(pool))
        revert InvalidPool();
    //existing code
}
```

C-03 Biased and Flawed Winner Selection Mechanism in RafflePool._fulfillTierPool

Severity **Critical** Status ● Resolved Fix commit 9cbc8e5

DESCRIPTION

The winner selection logic in _fulfillTierPool relies on taking each RNG value modulo ticketsSold and then probing linearly until an unused index is found:

```
uint256 randomIndex = rng[rngIndex] % ticketsSold;
uint256 attempts = 0;
uint256 selectedTicket = randomIndex;
while (_isTicketUsed(usedTickets, usedTicketsCount,
    selectedTicket) && attempts < ticketsSold) {
    selectedTicket = (selectedTicket + 1) % ticketsSold;
    attempts++;
}
```

This approach introduces a critical flaw. The modulus operation restricts all selections to the range $[0, \text{ticketsSold} - 1]$. However, ticket numbers are not guaranteed to fall within the sequential range $[0, \text{ticketsSold} - 1]$ — buyers may purchase any valid ticket number less than `totalTickets`.

EXAMPLE

- Consider a pool with `totalTickets = 1,000` and only `ticketsSold = 100`.
- If users purchased tickets numbered $[900-999]$

Those tickets would never be considered during winner selection because the algorithm only checks indices $[0-99]$.

As a result, only tickets with numbers < 100 are eligible to win.

This flaw creates a severe fairness issue: users holding higher-numbered tickets have zero chance of winning. In the extreme case where all sold tickets are numbered 100, every winning slot would resolve to

```
address(0), leaving prizes permanently unclaimable.
```

IMPACT

This issue undermines the fairness and reliability of the raffle system,

RECOMMENDED MITIGATION

You can revise the winner selection mechanisms to choose only from the actually sold tickets. A robust approach is to maintain an array or mapping of sold ticket numbers and select uniformly from this set using the index.

H-01 Absence of Robust Signature Replay Protection in `SignatureVerifier.verifySignature`

Severity High Status ● Resolved Fix commit `a47a075`

DESCRIPTION

The `verifySignature` function validates signatures but lacks proper replay protection. The signed payload does not incorporate safeguards such as the current `chainId`, leaving it exposed to cross-chain replay attacks. Additionally, the function currently uses the factory address as the `contractAddress` parameter rather than the pool's own address. Since the factory address remains constant across all pools, signatures are not uniquely bound to a specific pool. This design flaw allows a signature intended for one pool to be reused in another if the remaining arguments remain unchanged, making it impossible to reliably differentiate between pool-specific requests.

```
function verifySignature(
    bytes32 messageHash,
    bytes memory signature,
    address authorizedSigner,
    address contractAddress,
    address userAddress
) internal returns (bool) {
    bytes memory messageToHash = abi.encodePacked(
        userAddress,
        messageHash,
        @>> contractAddress
    );
    bytes32 ethSignedMessage = MessageHashUtils.toEthSignedMessageHash(
        messageToHash
    );
    address recovered = ethSignedMessage.recover(signature);
    if (recovered != authorizedSigner) revert InvalidSignature();
    emit SignatureVerified(userAddress, messageHash);
    return true;
}
```

IMPACT

Signatures can be replayed across pools and chains, potentially allowing unauthorized prize claims

RECOMMENDED MITIGATION

We recommend to implement the following changes :

In RafflePool.claimPrize() and RafflePool.buyTickets() :

```
bytes32 messageHash = SignatureVerifier.generateMessageHash(
    msg.sender,
    ILuckyBidsTypes.ContractAction.ClaimPrize
);
SignatureVerifier.verifySignature(
    messageHash,
    signature,
    IRaffleFactory(factory).authorizedSigner(),
    - factory,
    + address(this),
    msg.sender
);
```

In SignatureVerifier.verifySignature() :

```
+ uint private nonce;
function generateMessageHash(
    address user,
    ILuckyBidsTypes.ContractAction action
) internal pure returns (bytes32) {
-   return keccak256(abi.encodePacked(user, uint8(action)));
+   return keccak256(abi.encodePacked(user, uint8(action), block.chainId, nonce));
+   nonce++;
}
```

H-02 Incomplete Processing Fee Calculation in RafflePool.getTotalPrizeForUser and RafflePool.claim- Prize leads to revenue loss

Severity **High** Status ● Resolved Fix commit 9cbc8e5

DESCRIPTION

The getTotalPrizeForUser and claimPrize functions currently calculate the processing fee based solely on the prize amount of the highest winning category in which the user appears. Although the total prize is correctly aggregated across all categories and multiple wins, the processing fee is not applied proportionally to this aggregated total.

This results in an undercharged fee when a user wins multiple prizes across different categories or multiple times within a category.

```
function getTotalPrizeForUser(
    address user
) external view returns (uint256 totalPrize, uint256 processingFee, bool hasWon)
{
    if (state != ILuckyBidsTypes.PoolState.Settled) {
        return (0, 0, false);
    }
    uint256 totalPrizeAmount = 0;
    uint256 highestCategoryIndex = 4;
    for (uint256 i = 0; i < 4; i++) {
        for (uint256 j = 0; j < winnerCategories[i].winners.length; j++) {
            if (winnerCategories[i].winners[j] == user) {
                totalPrizeAmount += winnerCategories[i].prizePerWinner;
                if (i < highestCategoryIndex) {
                    highestCategoryIndex = i;
                }
            }
        }
    }
}
```

Example scenario:

Assume a bronze-tier pool with category prizes as follows: First = 1,500, Second = 400, Third = 100, Small = 20, and a processing fee of 100 bips (1%).

- User A wins one First prize, one Second prize, and five Third prizes.

- Total prize: $1,500 + 400 + (5 \cdot 100) = 2,400$.

Current implementation calculates the fee based only on the highest category prize: $1,500 \cdot 1\% = 15$. If the fee were applied proportionally to the full payout, it would be: $2,400 \cdot 1\% = 24$.

The user pays 15 instead of 24, resulting in a shortfall of 9 units for the protocol.

IMPACT

- The protocol collects lower fees than expected, reducing revenue.
- A user with multiple wins pays the same fee as a user with a single win in the highest category.

RECOMMENDED MITIGATION

Revise fee logic to apply processing fees to totalPrizeAmount, not just the highest category prize.

```
uint256 calculatedProcessingFee = 0;
if (
    highestCategoryIndex < 3 && winnerCategories[highestCategoryIndex].hasProcessingFee
) {
    IPoolTiers tierManager = IPoolTiers(tierParams.tierManager);
    IPoolTiers.TierConfig memory config = tierManager.getTierConfig(
        IPoolTiers.PoolTier(tierParams.poolTier)
    );
    uint256 highestCategoryPrize = winnerCategories
        [highestCategoryIndex].prizePerWinner;
    - calculatedProcessingFee = (highestCategoryPrize * config.processingFeeBps) / 10000;
    + calculatedProcessingFee = (totalPrizeAmount * config.processingFeeBps) / 10000;
}
```

M-01 Missing validation on prizeAmounts total in PoolTierManager.updateTierConfig

Severity **Medium** Status ● Resolved Fix commit d7bd38a

DESCRIPTION

The updateTierConfig function updates tier configuration values and includes checks on winner counts, array length, and processing fee. However, it does not validate that the sum of all prizeAmounts entries is 10,000.

Without this validation, the cumulative prize distribution can exceed 100% of the pool revenue. In such cases, when calculatePrizes is called, the computed payout amounts will be greater than the available revenue. In such a scenario, early winners who claim first may receive their full rewards, while later winners may be unable to claim due to pool funds being exhausted. This creates inconsistent payout behavior and a poor user experience.

IMPACT

Users who claim their prizes early can potentially drain the pool while leaving others unable to claim their prizes.

RECOMMENDED MITIGATION

Introduce a validation step in `updateTierConfig` that checks:

```
if (config.firstPlaceWinners != 1) revert
FirstPlaceMustHaveOneWinner();
if (config.secondPlaceWinners < 2
    || config.secondPlaceWinners > 4) revert InvalidSecondPlaceWinnersCount();
if (config.thirdPlaceWinners < 5 || config.thirdPlaceWinners > 20) revert
InvalidThirdPlaceWinnersCount();
if (config.smallWinsCount < 50 || config.smallWinsCount > 170) revert InvalidSmallWinsCount();
if (config.processingFeeBps > 1000) revert ProcessingFeeExceedsLimit();
if (config.prizeAmounts.length != 4) revert MustHaveFourPrizeAmounts();
+ uint256 totalPrize;
+ for (uint256 i; i < 4; i++) {
+     totalPrizeBps += config.prizeAmounts[i];
+ }
+ if (totalPrize >= 10000) revert InvalidPrizeAmounts();
```

M-02 Pool ownership tracking broken in `RaffleFactory.createPool` function

Severity **Medium** Status ● Resolved Fix commit 6ae74de

DESCRIPTION

In `createPool`, after deploying a new pool, the contract records necessary info such as the creator and user-owned pool indexes. However, instead of using the `_poolCreator` argument, the function incorrectly uses `msg.sender` (the factory owner, enforced by `onlyOwner`). As a result, all pools are attributed to the factory owner rather than the intended creator.

This misattribution causes accounting inconsistencies: queries such as `getUserPools` or `getUserPoolsPaginated` will always return zero for actual pool creators, rendering these functions ineffective for their intended purpose.

```
uint256 userPoolIndex = userPoolCount[msg.sender];
userPoolAtIndex[msg.sender][userPoolIndex] = poolAddress;
userPoolCount[msg.sender]++;
emit PoolCreated(
    poolAddress,
    @>> msg.sender,
    params.totalTickets,
    params.ticketPrice,
    params.poolTier,
    params.platformFeeBps,
    params.paymentToken
);
```

IMPACT

All pools appear as if they were created by the factory owner.

RECOMMENDED MITIGATION

Replace all uses of `msg.sender` in the pool attribution logic with `_poolCreator`

```
- uint256 userPoolIndex = userPoolCount[msg.sender];
+ uint256 userPoolIndex = userPoolCount[_poolCreator];
- userPoolAtIndex[msg.sender][userPoolIndex] = poolAddress;
+ userPoolAtIndex[_poolCreator][userPoolIndex] = poolAddress;
- userPoolCount[msg.sender]++;
+ userPoolCount[_poolCreator]++;
emit PoolCreated(
    poolAddress,
-   msg.sender,
+   _poolCreator,
    params.totalTickets,
    params.ticketPrice,
    params.poolTier,
    params.platformFeeBps,
    params.paymentToken
);
```

L-01 Missing validation on `maxTicketsPerAddress` relative to `totalTickets` in `RaffleFactory._validatePoolParams`

Severity **Low** Status ● Resolved Fix commit `6ae74de`

DESCRIPTION

The `_validatePoolParams` function enforces several constraints on `TierPoolParams`, such as ensuring nonzero tickets, positive ticket price, correct sale and settlement timings, and fee limits. However, it does not check that `maxTicketsPerAddress` is strictly less than `totalTickets`. Without this validation, a single participant could purchase all available tickets in a pool. While the contracts payout logic and fee deduction may still technically work, this undermines the fairness of the raffle mechanism and creates a poor user experience.

IMPACT

A single address can monopolize the raffle, making it non-competitive. Making system unfair and misleading

RECOMMENDED MITIGATION

Add a validation step in `_validatePoolParams` to ensure :

```
{ if (params.totalTickets == 0) revert InvalidPoolParams();
  if (params.ticketPrice == 0) revert InvalidPoolParams();
  if (params.saleStart >= params.saleEnd) revert InvalidPoolParams();
  if (params.saleEnd >= params.settlementTime) revert InvalidPoolParams();
  if (params.settlementTime >= params.endTime) revert InvalidPoolParams();
  if (params.platformFeeBps > 10000) revert InvalidPoolParams();
  if (params.maxTicketsPerAddress == 0) revert InvalidPoolParams();
  if (params.poolTier > 2) revert InvalidPoolParams();
  if (params.tierManager == address(0)) revert InvalidPoolParams();
+  if (params.maxTicketsPerAddress >= params.totalTickets) revert InvalidPoolParams();
}
```

L-02 Missing check whether `_owner` is wallet address or not in `RaffleVrfRouter` contract

Severity **Low** Status ● Resolved Fix commit 2f68dbf

DESCRIPTION

The `RaffleVrfRouter` contract relies on `_owner` as the `_clientWalletAddress` when calling

`Supras.generateRequest`. Supra requires this argument to be a valid whitelisted wallet address. However, during initialization the contract does not verify whether the provided owner address is an EOA (wallet) or a contract.

IMPACT

If a contract address is set as the owner, randomness requests will fail since contracts cannot act as whitelisted wallets on Supra. This results in a denial-of-service for all RNG requests.

RECOMMENDED MITIGATION

Add a validation step in the initializer:

```
+ error InvalidOwner();
function initialize(
    address _supraRouter,
    address _owner
) public initializer {
+   if (_owner == address(0)
        || _owner.code.length > 0) revert InvalidOwner();
    __Ownable_init(_owner);
    __UUPSUpgradeable_init();
    supraRouter = _supraRouter;
    active = true;
}
```

L-03 Missing validation on totalTickets in RaffleFactory._validatePoolParams

Severity **Low** Status ● Resolved Fix commit 6ae74de

DESCRIPTION

The `_validatePoolParams` function ensures that pool parameters such as ticket price, time windows, and per-address limits are valid. However, it does not enforce a minimum bound on `totalTickets`. This becomes problematic because `PoolTierManager.updateTierConfig` hardcodes the number of winners to be between 58 and 195. If `totalTickets` is configured below the minimum winner count (e.g., 50 tickets), the system ends up in an inconsistent state where it is mathematically impossible to select the required number of winners.

RECOMMENDED MITIGATION

Add an explicit validation in `_validatePoolParams` to ensure that `totalTickets` is always greater than or equal to the minimum winner count:

```
+ if (params.totalTickets < MIN_WINNER_COUNT) revert InvalidPoolParams(); //MIN_WINNER_COUNT = 58
```

L-04 Duplicate Ticket Numbers in Single Purchase Cause ticketsSold Accounting Discrepancy in Raffle- Pool.buyTickets

Severity **Low** Status ● Resolved Fix commit 9cbc8e5

DESCRIPTION

The `buyTickets` function verifies that each requested ticket number is within range and not already sold by checking `ticketOwner[ticketNumber]`. However, it does not enforce that the `ticketNumbers` array contains unique values within a single transaction.

Now even in such case user have to pay the fee for the `ticketNumber.length` making protocol profitable, but it would cause limited possibility for other users to become eligible for winning.

As a result, a user can submit duplicate entries in the same purchase request, such as

`[16,16,16,16,16,16,16]`.

In this case, only one `ticketNumber = 16` is assigned to the caller, but the contract increments `ticketsSold` and `userTicketCount` by 7.

This causes a discrepancy between the `ticketsSold` counter and the actual number of unique tickets sold. It reduces the effective availability of tickets for other participants and can confuse off-chain accounting that rely on `ticketsSold` being accurate. While the protocol gains extra payment from the user in this scenario, the overall fairness of the raffle is compromised.

IMPACT

`ticketsSold` and `userTicketCount` become inflated relative to actual unique ticket ownership. This would reduce the fairness

RECOMMENDED MITIGATION

Add validation to ensure the ticketNumbers array contains unique values before processing purchases.:

```
for (uint256 i = 0; i < ticketNumbers.length; i++) {
    uint256 ticketNumber = ticketNumbers[i];
    if (ticketNumber >= tierParams.totalTickets) {
        revert TicketNumberOutOfRange();
    }
    if (ticketOwner[ticketNumber] != address(0)) {
        revert TicketAlreadySold();
    }
    + ticketOwner[ticketNumber] = msg.sender;
    + userTickets[msg.sender].push(ticketNumber);
}
uint256 totalCost = ticketNumbers.length * tierParams.ticketPrice;
if (tierParams.paymentToken == address(0)) {
    if (msg.value != totalCost) revert InsufficientPayment();
} else {
    if (msg.value != 0) revert InsufficientPayment();
    IERC20(tierParams.paymentToken).safeTransferFrom(
        msg.sender,
        address(this),
        totalCost
    );
}
// Assign tickets to the buyer
- for (uint256 i = 0; i < ticketNumbers.length; i++) {
-     uint256 ticketNumber = ticketNumbers[i];
-     ticketOwner[ticketNumber] = msg.sender;
-     userTickets[msg.sender].push(ticketNumber);
- }
ticketsSold += ticketNumbers.length;
userTicketCount[msg.sender] += ticketNumbers.length;
```

L-05 False Event Emission in RafflePool._fulfillTierPool

Severity Low Status ● Resolved Fix commit 9cbc8e5

DESCRIPTION

The `_fulfillTierPool` function is the core settlement function of the raffle. It determines the number of winners per category, calculates prize amounts, and updates state accordingly. Once all internal state updates are complete, it emits the `WinnersSelected` event.

However, instead of emitting the actual winners, ticket numbers, and prizes, the function currently emits empty arrays. This creates a discrepancy between the true on-chain state and the information made available through events. Off-chain services that rely on these events for updates will therefore receive incomplete or misleading information.

As a result, users and external systems may believe that no winners were selected even though the contract has already finalized winners and prize allocations.

IMPACT

This would cause incorrect off-chain state and transparency issue.

RECOMMENDED MITIGATION

Consider emitting event with correct info

L-06 Missing endTime Check Before Drawing in RafflePool.requestDraw

Severity Low Status Resolved Fix commit 67a53b1

DESCRIPTION

In the requestDraw function, the contract transitions the pool state from Live to Drawing and submits a VRF request for randomness. However, the current implementation only checks that:

- The pool is in Live state,
- The settlementTime has passed, and
- At least one ticket has been sold.

However, it does not validate that the current block timestamp is less than or equal to endTime. This means that the pool owner can still trigger a draw even after the pool has passed its intended expiration point.

IMPACT

Expired pools can still settle leading to inconsistent lifecycle enforcement

RECOMMENDED MITIGATION

Add an explicit check to enforce the pools expiry in requestDraw::

```
function requestDraw() external onlyOwner whenPoolNotPaused {
    if (state != ILuckyBidsTypes.PoolState.Live) revert PoolNotLive();
    - if (block.timestamp < tierParams.settlementTime)
    + if (block.timestamp < tierParams.settlementTime || block.timestamp > tierParams.endTime)
      revert PoolNotSettled();
    if (ticketsSold == 0) revert NoTicketsSold();
}
```

L-07 Potential Out-Of-Gas Risk in RafflePool._fulfillTierPool

Severity Low Status Resolved Fix commit 9cbc8e5

DESCRIPTION

The _fulfillTierPool is responsible for selecting raffle winners once randomness is provided. While functionally correct in its intent, its current implementation poses serious gas efficiency concerns and carries a high risk of out-of-gas (OOG) failures under realistic operating conditions, especially if the winners-to-ticketsSold ratio is very low.

The problem lies in the winner selection algorithm, which relies on nested loops for each winner. And when a randomly selected ticket is already taken, the function performs a linear search using `while()` loop to find the next available ticket.

In the best-case scenario, the function is unlikely to run out of gas when selecting up to 195 winners. However, in a worst-case scenario—for example, when `ticketsSold` reaches 10,000—the repeated state reads and writes across nested loops can result in extremely high gas consumption.

IMPACT

The function may exhaust the block gas limit, causing the transaction to revert and preventing the winner selection process from completing.

RECOMMENDED MITIGATION

We recommend these two approaches:

On-chain: Maintain an array of sold tickets and use an $O(1)$ swap-and-pop selection algorithm to ensure scalability and fairness.

Hybrid off-chain: Perform the gas-intensive winner computation off-chain using the on-chain randomness seed, then commit the results on-chain

I-01 Ineffective onlyValidFactory Modifier

Severity **Informational** Status ● Resolved Fix commit 9cbc8e5

DESCRIPTION

The `onlyValidFactory` modifier is defined as:

```
modifier onlyValidFactory() {
  @> if (factory == address(0)) revert ZeroAddress();
  _;
}
```

This check only ensures that the factory address is non-zero. It does not verify that the caller is the factory (`msg.sender == factory`). As a result, any account can call functions guarded by this modifier as long as factory has been set, which defeats the purpose of access control.

Although the modifier is not currently applied to any function, its presence suggests it may be used in future upgrades. If applied without correction, it would provide no real protection

I-02 Lack of Ticket Availability Function in RafflePool Causes Failed Purchases and Wasted Gas

Severity **Informational** Status ● Acknowledged

DESCRIPTION

In the current implementation of `buyTickets`, users are required to provide an array of ticket- Numbers they wish to purchase. However, if even a single ticket number in the array has already been sold, the entire transaction reverts with `TicketAlreadySold()`.

This design can be particularly frustrating in high-demand scenarios where ticket availability changes quickly. Moreover, contract lacks view function for users to easily check ticket availability in bulk; the only option is to query `ticketOwner[ticketNumber]` individually for each ticket. This creates inefficiency and a poor user experience.

Introducing a dedicated view function to return the availability of one or multiple ticket numbers would allow users to pre-check availability before executing a purchase, thereby saving gas and improving the overall user experience.

RECOMMENDED MITIGATION

You can implement a function like this:

```
function isTicketSold(uint256 ticketNumber) external view returns (bool) {
    return ticketOwner[ticketNumber] != address(0);
}
function areTicketsSold(uint256[] calldata ticketNumbers)
external
view
returns (bool[] memory)
{
    bool[] memory result = new bool[](ticketNumbers.length);
    for (uint256 i; i < ticketNumbers.length; i++) {
        result[i] = ticketOwner[ticketNumbers[i]] != address(0);
    }
    return result;
}
```

I-03 Incomplete Validation of ticketPrice in RaffleFactory._validatePoolParams

Severity **Informational** Status ● Acknowledged

DESCRIPTION

The function `_validatePoolParams` validates pool parameters before deployment. While it ensures that `ticketPrice` is not zero, it does not impose any upper bound. This omission allows the creation of pools with excessively high or unrealistic ticket prices.

IMPACT

Pools with abnormally high `ticketPrice` values may become unusable for participants.

RECOMMENDED MITIGATION

Introduce a `MAX_TICKET_PRICE` constant and enforce it during validation:

```
uint256 constant MAX_TICKET_PRICE = 1000e6; //if paymentToken is USDC
if (params.ticketPrice == 0 || params.ticketPrice > MAX_TICKET_PRICE) {
    revert InvalidPoolParams();
}
```

I-04 Use Ownable2StepUpgradeable instead of OwnableUpgradeable

Severity **Informational** Status ● Acknowledged

DESCRIPTION

The `Ownable2StepUpgradeable` prevents the contract ownership from mistakenly being transferred to an address that cannot handle it, by requiring that the recipient of the owner's permissions actively accept via a contract call of its own.

RECOMMENDED MITIGATION

```
- import {OwnableUpgradeable} from "@openzeppelin/contracts-
upgradeable/access/OwnableUpgradeable.sol";
+ import {Ownable2StepUpgradeable} from "@openzeppelin/contracts-
upgradeable/access/Ownable2StepUpgradeable.sol";
```

I-05 Few Important State Change Lacks Event Emission

Severity **Informational** Status ● Acknowledged

DESCRIPTION

Consider emitting event for necessary state changes

```
function setPaymentToken(address _paymentToken) external onlyOwner {...}
function setDefaultPlatformFeeBps(...)
function setDefaultMaxTicketsPerAddress(...)
function cancelPool() external onlyOwner {...}
function updatePoolState(...)
```

I-06 RaffleVrfRouter.myCallback does not follow the CEI pattern

Severity **Informational** Status ● Resolved Fix commit 2f68dbf

DESCRIPTION

The myCallback function can only be accessed by supraRouter. However, it performs an external call to the pool contract (pool.call) before updating its internal state mappings (nonceToPool, nonceToRequestId).

This violates the Checks-Effects-Interactions (CEI) pattern, where state changes should be made before any external interactions.

Consider implementing these changes:

```
function myCallback(uint256 nonce, uint256[] calldata rngList) external {
    if (msg.sender != supraRouter) revert OnlySupraRouterCanCall();
    address pool = nonceToPool[nonce];
    if (pool == address(0)) revert InvalidRequest();
    uint256 requestId = nonceToRequestId[nonce];
+   delete nonceToPool[nonce];
+   delete nonceToRequestId[nonce];
    (bool success, ) = pool.call(
        abi.encodeWithSignature(
            "fulfillRandomness(uint256,uint256[])",
            requestId,
            rngList
        )
    );
    if (!success) revert VrfCallbackFailed();
    emit RngFulfilled(pool, requestId, nonce, true);
-   delete nonceToPool[nonce];
-   delete nonceToRequestId[nonce];
}
```

I-07 No Refund Mechanism for Users

Severity Informational Status ● Acknowledged

DESCRIPTION

The RafflePool contract allows users to purchase tickets using ETH or ERC-20 tokens through the buyTickets function. If the pool is cancelled the contract sets the pools state to Cancelled prior to any Draw. However, there is no mechanism for users to recover their ticket payments. Instead, the only way to withdraw funds is through emergencyWithdraw(), which is restricted to the contract owner. As a result, users who have purchased tickets cannot reclaim their funds, effectively losing their payments if the pool is cancelled.

RECOMMENDED MITIGATION

Implement a user refund mechanism to allow participants to reclaim their ticket payments atleast in the event of a pool cancellation

G-01 Avoid redundant logic and unnecessary state variables to reduce gas

Severity Informational Status ● Resolved Fix commit 9cbc8e5 Category Gas

DESCRIPTION

Several parts of the contract contain redundant logic and unnecessary state variables. Removing these not only simplifies the codebase but also reduces both deployment and runtime gas costs. - (

```
- uint256 firstCount,
- uint256 secondCount,
- uint256 thirdCount,
- uint256 smallCount
- ) = tierManager.getWinnerCounts(
- IPoolTiers.PoolTier(tierParams.poolTier)
- );
- uint8 totalRngCount = uint8(
- firstCount + secondCount + thirdCount + smallCount
- );
+ uint8 totalRngCount = tierManager.getTotalWinners( IPoolTiers.PoolTier(tierParams.poolTier) );
- while (_isTicketUsed(usedTickets, usedTicketsCount, selectedTicket) && attempts <
- ticketsSold) {...}
+ while (_isTicketUsed(usedTickets, usedTickets.length, selectedTicket) && attempts <
- ticketsSold) {...}
```

The logic in RafflePool.claimPrize can be simplified and refactored as follows:

```
function getTotalPrizeForUser(
    address user
)
- external
+ public
view
returns (uint256 totalPrize, uint256 processingFee, bool hasWon)
```

```

{
    function claimPrize(bytes calldata signature) external nonReentrant {
        if (state != ILuckyBidsTypes.PoolState.Settled) revert PoolNotLive();
        if (hasClaimedPrize[msg.sender]) revert AlreadyClaimed();
        bytes32 messageHash = SignatureVerifier.generateMessageHash(
            msg.sender,
            ILuckyBidsTypes.ContractAction.ClaimPrize
        );
        SignatureVerifier.verifySignature(
            messageHash,
            signature,
            IRaffleFactory(factory).authorizedSigner(),
            factory,
            msg.sender
        );
+       (uint256 totalPrizeAmount, uint256 totalProcessingFee, ) =
        getTotalPrizeForUser(msg.sender);
        if (totalPrizeAmount == 0) revert NotAWinner();
        // Mark as claimed
        hasClaimedPrize[msg.sender] = true;
        totalPrizeAmount -= totalProcessingFee;
        // existing code
    }
}

```

In RafflePool._fulfillTierPool, the firstWinnerCount is always set to 1 in the PoolTierManager (if (config.firstPlaceWinners != 1) revert FirstPlaceMustHaveOneWinner();). Therefore, the surrounding logic is redundant and can be safely removed:

```

// First Place Winners - Limited by available tickets
- uint256 actualFirstCount = ticketsSold > 0 ? min(firstCount, ticketsSold) : 0;
+ uint256 actualFirstCount = 1;
winnerCategories[0].winners = new address[](actualFirstCount);
- winnerCategories[0].prizePerWinner = actualFirstCount > 0 ? prizes[0] : 0;
+ winnerCategories[0].prizePerWinner = prizes[0];
- winnerCategories[0].totalPrize = actualFirstCount > 0 ? prizes[0] * actualFirstCount
: 0;
+ winnerCategories[0].totalPrize = prizes[0];
- winnerCategories[0].hasProcessingFee = true; //not required already set during initialization
- for (uint256 i = 0; i < actualFirstCount; i++) {
    uint256 randomIndex = rng[rngIndex] % ticketsSold;
    uint256 attempts = 0;
    uint256 selectedTicket = randomIndex;
    while (_isTicketUsed(usedTickets, usedTicketsCount, selectedTicket) && attempts <
        ticketsSold) {
        selectedTicket = (selectedTicket + 1) % ticketsSold;
        attempts++;
    }
    usedTickets[usedTicketsCount] = selectedTicket;
    usedTicketsCount++;
    address winner = ticketOwner[selectedTicket];
    winnerCategories[0].winners[i] = winner;
    rngIndex++;
- }

```

G-02 Unused Error

Severity Informational

Status ● Resolved Fix commit 67a53b1

Category Gas

DESCRIPTION

- Consider removing the unused error to save gas.
- Found in IRaffleFactory.sol

```
- error InvalidPaymentToken();  
- error AlreadyDrawn();  
- error NotEnoughRandomness();  
- error RngRequestTimedOut();  
- error InvalidPoolParams();
```

G-03 Dead Code

Severity Informational

Status ● Resolved Fix commit 67a53b1

Category Gas

DESCRIPTION

Consider removing functions that are not used.

```
- function _isAddressInArray(...)  
- modifier onlyValidFactory() {  
-   if (factory == address(0)) revert ZeroAddress();  
-   _;  
- }
```

Disclaimer

This report is based on the code at the commit listed in the Scope section and on information the client provided during the review. It reflects Safe Edges' assessment at the time of the review and is not a guarantee that the code is free of vulnerabilities.

Changes made after the reviewed commit, deployment configuration, off-chain components and third-party dependencies are outside the scope of this report unless stated otherwise.

Security is ongoing. We recommend continued testing, monitoring and bug bounty coverage alongside any audit.

This report is not investment advice and is not an endorsement of the project, its team or any token.



Safe Edges is a Web3 security firm and official security partner for the Fuel, ENI and XI EcoChain ecosystems. We work across smart contract audits, Rust and WASM reviews, ZK circuit review, AI and MCP security, and penetration testing.

200+

PROJECTS AUDITED

\$4B+

TVL SECURED

0

POST-AUDIT HACKS