RushWin Security Audit

Report Version 1.0

April 30, 2024

Conducted by **Hunter Security**:

BountyHunt3r, Senior Security Auditor
Stormy, Associate Security Auditor

Table of Contents

1 About Hunter Security

Hunter Security consists of multiple teams of leading smart contract security researchers. Having conducted over 100 security reviews and reported tens of live smart contract security vulnerabilities protecting over $1B of TVL, our team always strives to deliver top-quality security services to DeFi protocols. For security audit inquiries, you can reach out to us on Telegram or Twitter at *@georgehntr*.

2 Disclaimer

Audits are a time-, resource-, and expertise-bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can reveal the presence of vulnerabilities **but cannot guarantee their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - involves a small loss of funds or affects a core functionality of the protocol.
- **Low** - encompasses any unexpected behavior that is non-critical.

3.2 Likelihood

- **High** - a direct attack vector; the cost is relatively low compared to the potential loss of funds.
- **Medium** - only a conditionally incentivized attack vector, with a moderate likelihood.
- **Low** - involves too many or unlikely assumptions; offers little to no incentive.

3.3 Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Hunter Security was engaged by RushWin to review their smart contract protocol during the period from April 24, 2024, to April 30, 2024.

Overview

Project Name	RushWin
Repository	https://github.com/rush-win/rush.win-contracts-beta
Commit hash	4e6133d216a3257168f1a768f71c0fe9c51ba534
Resolution	fe16d28e16d8da1e1e8b0d8b1ae348bb1962c80f
Methods	Manual review & testing

Timeline

-	April 24, 2024	Audit kick-off
v0.1	April 30, 2024	Preliminary report
v1.0	April 30, 2024	Mitigation review

Scope

contracts/Rush.sol
contracts/Referral.sol
contracts/lib/ReferralMap.sol

Issues Found

High risk	0
Medium risk	3
Low risk	0

5 Findings

5.1 Medium

5.1.1 Error between the max reserved amount for payouts and the actual amount payed on winning

Severity: Medium

Description: The system fetches the value of *worstCasePayout* based on the *maxMultiplier*.

```
uint256 worstCasePayout = (totalAmountActiveHypercubes *
    maxMultiplier) / BASIS;
```

However, when a hypercube wins, the actual amount paid is *multiplier - x1* as the payout includes the initial bet amount which isn't counted in the house balance in the first place.

```
uint256 payout = (amount * multiplier) / BASIS;
house = house + amount - payout;
```

If we take the below example of *maxMultiplier*, when the *worstCasePayout* value is calculated the system will reserve amount of *totalAmountActiveHypercubes * 6* in order to pay all active cubes in the worst possible outcome. However, if we take the fact that this max multiplier holds the initial bets of all cubes which aren't counted in the house balance the needed amount to pay all cubes would actually be *totalAmountActiveHypercubes * 5*.

```
maxMultiplier = 6 * BASIS;
```

Recommendation: Consider implementing the following change:

```
uint256 worstCasePayout = (totalAmountActiveHypercubes * (maxMultiplier - BASIS)) /
    BASIS;
```

Resolution: Resolved.

5.1.2 Worst case scenario is wrongly calculated

Severity: Medium

Description: The system calculates the amount of the worst payout the house owns based on the max multiplier and the *totalAmountActiveHypercubes*.

```
uint256 worstCasePayout = (totalAmountActiveHypercubes *
    maxMultiplier) / BASIS;
```

The problem here is that on every reveal the system gives out a commission to the referrals which is also paid out from the house balance, while this is not taken in fact when calculating the *worstCasePayout*.

```
uint256 comission = (amount * REFERRER_SHARE) / BASIS;
if (_referrers.addCommission(user, comission)) {
    // Update state
    house -= comission;
}
```

Recommendation: Consider paying out the commission on hypercube buy instead of hypercube reveal and updating the house balance.

Resolution: Resolved.

5.1.3 The function withdrawRandomizer is temporary DoS-ed

Severity: Medium

Description: The function *withdrawRandomizer* doesn't allow any deposit to be withdrawn from the randomizer when there is a positive number of active cubes.

```
function withdrawRandomizer() external onlyOwner whenPaused {
    require(totalNumberActiveHypercubes == 0, "hypercubes active");
    (uint256 deposit, uint256 reserved) = _randomizer.clientBalanceOf(
        address(this)
    );
    _randomizer.clientWithdrawTo(owner(), deposit - reserved);
}
```

However, when buying a hypercube the system ensures that there is enough deposited to cover the gas, which in fact is later added to the reserved amount and subtracted from the deposit one when calling the randomizer and making the request.

```
// Top up VRF if needed
(uint256 deposit, uint256 reserved) = _randomizer.clientBalanceOf(
    address(this)
);
uint256 freeDeposit = deposit - reserved;

// Add a buffer to the estimated fee
uint256 estimatedFee = (_randomizer.estimateFee(CALLBACK_GAS_LIMIT) *
    CALLBACK_GAS_BUFFER) / BASIS;

if (freeDeposit < estimatedFee) {
    uint256 addToDeposit = estimatedFee - freeDeposit;
    _randomizer.clientDeposit{value: addToDeposit}(address(this));

    house -= addToDeposit;
}
```

```
// Request a random number from randomizer
uint256 nonce = _randomizer.request(CALLBACK_GAS_LIMIT);
```

Recommendation: Consider removing the check.

Resolution: Resolved.