Base Flip Security Audit

Report Version 1.1

April 30, 2024

Conducted by **Hunter Security**:

BountyHunt3r, Senior Security Researcher
Stormy, Security Researcher
George Hunter, Peer Reviewer

Table of Contents

1 About Hunter Security

Hunter Security is a team of independent smart contract security researchers. Having conducted over 100 security audits 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

The Hunter Security team was engaged by Base Flip to conduct a security audit of their betting protocol during the period from March 29th, 2024, to April 2nd, 2024.

Overview

Project Name	Base Flip
Repository	https://github.com/dragon-software-devs/BaseFlip
Commit hash	4945a6ccb42b574c37cab193686d1ee88326c45a
Resolution	c90e9dd78150c7640ebe2654811940ef944205d0
Update review	f7b9744fd540b94bc72413dc96764bead0fb8a8c
Methods	Manual review

Timeline

-	March 29, 2024	Audit kick-off
v0.1	April 2, 2024	Preliminary report
v1.0	April 3, 2024	Mitigation review
v1.1	April 30, 2024	Update review

Scope

contracts/Flip.sol
contracts/lib/ReferralMap.sol

Issues Found

High risk	2
Medium risk	2
Low risk	2

5 Findings

5.1 High

5.1.1 Revealed bets are always eligible for recovery

Severity: *High*

Context: Flip.sol#L434-L450

Description: The randomizer callback function is used to provide an outcome for a bet. Once this happens the system marks the bet as revealed, and the player either wins or loses the bet.

```
bet.totalPayout = totalPayout;
bet.probability = probability;
bet.revealed = true;

// Update user info
userInfo.totalPayout += totalPayout;
emit BetRevealed(user, id, totalPayout);
```

The function *recoverBet()* is used by the user to recover their initial bet if the callback function was not executed within the *bet.timeout* duration.

```
function recoverBet(uint256 id) external {
    Bet storage bet = bets[id];
    address user = bet.user;

    require(user == msg.sender, "unauthorized");
    require(block.timestamp > bet.timeout, "active");

    // Refund
    Address.sendValue(payable(user), bet.betAmount);

    // Mark bet as recovered
    bet.recovered = true;

    // Update state
    totalAmountActiveBets -= bet.betAmount;
    totalNumberActiveBets--;
}
```

The problem is that *recoverBet()* does not check if the bet has already been *recovered* or if it has been *revealed*. Therefore, two critical issues arise:

1. If the player wins or loses and their bet is marked as *revealed* they can recover their bet.
2. If the player recovered their bet once, and the bet is marked as *recovered*, they can still recover it again, effectively draining the contract.

Recommendation: Consider implementing the following checks in *recoverBet()*:

```
require(!bet.recovered, "bet recovered");
require(!bet.revealed, "already revealed");
```

Resolution: Resolved. The recommended fix was implemented.

5.1.2 Multiple re-entrancy attack vectors

Severity: *High*

Context: Flip.sol#L413, Flip.sol#L442

Description: The code exhibits multiple violations of the CEI Pattern which introduce multiple re-entrancy attacks. This is also aggravated by not using a re-entrancy part. Some instances:

1. An attacker can re-enter the *recoverBet()* function and drain the contract before the bet status is updated.
2. In case a user wins a bet and the *bet.timeout* has been exceeded, an attacker can re-enter into the *recoverBet()* function when the callback function attempts to send the winning amount. This would allow the attacker to withdraw their *bet.amount* in addition to getting *bet.amount * 2* for winning.

Recommendation: It is recommended to always use a re-entrancy guards, as well as adhere to the CEI pattern throughout the code.

Resolution: Resolved. The recommended fix was implemented.

5.2 Medium

5.2.1 Silent loss of commission when transferring referrer id

Severity: *Medium*

Context: ReferralMap.sol#L138-L157

Description: The commission logic in *ReferralMap* depends on the *storedId*, which is retrieved by calling *map._referrerToId[referrer]* where *referrer* is the address to get the commissions for.

The function *transferId()* allows to transfer the *storedId* from one address to another. This will subsequently move all the commissions to that new address, without claiming them first to the original owner.

Ultimately, the commissions will only be claimable by the new address.

Recommendation: It is recommended to claim the commissions first, if any, to the original owner, before doing the transfer.

Resolution: Acknowledged. The client stated that not claiming the commissions is the intended behavior of this functionality.

5.2.2 Incorrect reserved balance is assumed when resetting the house

Severity: *Medium*

Context: Flip.sol#L493

Description: The function *resetHouse()* allows the contract owner to reduce the house balance to a given amount.

The system calculates the reserved balance based on the total amount of active bets times two. However, this is not correct as the house is not responsible for paying double the bet amount, in the worst case scenario. But for any given bet of *x* amount, if the user wins they get an amount of *2x* and the house only pays *x*, while the other *x* comes from the initial bet amount deposited by the user. In light of this, the system only needs to reserve *totalAmountActiveBets* instead of *totalAmountActiveBets * 2*.

Recommendation: Use a value of *totalAmountActiveBets* for the reserved amount instead of *totalAmountActiveBets * 2*.

Resolution: Resolved. The recommended fix was implemented.

5.3 Low

5.3.1 Wrong maximum liability check when placing a bet

Severity: *Low*

Context: Flip.sol#L298

Description: In the function *placeBet()* the system checks if the house balance is not enough to compensate all the active bets. In that case the system will not accept any bets until the owner tops up the house balance. As mentioned in the previous issue, the house balance does not need to hold an amount of double the active bets, it only needs to hold *totalAmountActiveBets* in the worst case scenario.

Recommendation: It is recommended to review this logic, and modify it, so the house only needs to hold _'*totalAmountActiveBets* in the worst case scenario.

Resolution: Resolved. The recommended fix was implemented.

5.3.2 The owner can front-run a winning bet and withdraw the house balance

Severity: *Low*

Context: Flip.sol#L520-L526

Description: The function *withdrawHouse()* allows the owner to withdraw the entire house balance when the contract is paused, and reset it to 0. This will prevent any winning bet from being paid because all wins are paid from the house balance.

Furthermore, the owner can just keep observing the mempool until they see a callback with a winning number, and then front-run this transaction to pause and withdraw the house balance, effectively denying payment of rewards.

Recommendation: It is recommended to only allow the withdrawal of a portion of the house balance that is considered a surplus. This portion can be calculated through the following formula:

```
surplus = house - totalAmountActiveBets
```

This ensures that the amount remaining in the contract is still enough to cover all the bets in the worst-case scenario.

Resolution: Resolved. The recommended fix was implemented.