SB SECURITY

# Uranus

# Security Review


Feb 1, 2025

Conducted by:
**Blckhv**, Lead Security Researcher
**Slavcheww**, Lead Security Researcher

# Contents

# 1. About SBSecurity

**SBSecurity** is a duo of skilled smart contract security researchers. Based on the audits conducted and numerous vulnerabilities reported, we strive to provide the absolute best security service and client satisfaction. While it's understood that 100% security and bug-free code cannot be guaranteed by anyone, we are committed to giving our utmost to provide the best possible outcome for you and your product.

Book a Security Review with us at sbsecurity.net or reach out on Twitter @Slavcheww.

# 2. Disclaimer

A smart contract security review can only show the presence of vulnerabilities **but not their absence**. Audits are a time, resource, and expertise-bound effort where skilled technicians evaluate the codebase and their dependencies using various techniques to find as many flaws as possible and suggest security-related improvements. We as a company stand behind our brand and the level of service that is provided but also recommend subsequent security reviews, on-chain monitoring, and high whitehat incentivization.

# 3. Risk classification

| | Impact: High | Impact: Medium | Impact: Low |
|---|---|---|---|
| Likelihood: High | Critical | 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** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality.
- **Low** - funds are not at risk.

## 3.2. Likelihood

- **High** - almost **certain** to happen, easy to perform, or highly incentivized.
- **Medium** - only **conditionally possible**, but still relatively likely.
- **Low** - requires specific state or **little-to-no incentive**.

## 3.3. Action required for severity levels

- High - **Must** fix (before deployment if not already deployed).
- Medium - **Should** fix.
- Low - **Could** fix.

# 4. Executive Summary

### Overview

| | |
|---|---|
| Project | Uranus |
| Repository | Private |
| Commit Hash | 341aa9fa0b8b324e78948fa0031a2fa1e4683caa |
| Resolution | acb7a5e38fcac39e7f3e1cfd218bfda5b99a025d |
| Timeline | February 1, 2025 |

### Scope

| |
|---|
| Uranus.sol |
| UranusBuyBurn.sol |
| DevDistribute.sol |

### Issues Found

| | |
|---|---|
| Critical Risk | 1 |
| High Risk | 0 |
| Medium Risk | 0 |
| Low/Info Risk | 6 |

# 5. Findings

## 5.1. Critical severity

### 5.1.1. `rValue` is passed to `_rBurn()` instead `tValue`

**Severity:** Critical Risk

**Description:** In `distributeReserve()`, when part of the allocation is burned because the whole function uses `rValues`, it passes the allocation to burn in `rValue`, but `_rBurn()` assumes it's in `tValue` and internally converts it back to `rValue`. Because of this, accounting will be disrupted.

```
function distributeReserve() external {
    if (!tradingEnabled) revert TradingDisabled();
    uint256 rBalance = _rOwned[address(this)];
    if (rBalance == 0) revert InsuffucientBalance();

    uint256 rIncentive = FullMath.mulDiv(rBalance, incentiveFee, BPS_BASE);
    _rDistribute(msg.sender, rIncentive);
    uint256 rDistribution = rBalance - rIncentive;

    uint256 burnShare = rDistribution / RESERVE_DIVISOR;
    _rBurn(address(this), burnShare); <-----------------------------------
    rDistribution -= burnShare;

    uint256 lpShare = rDistribution / RESERVE_DIVISOR;
    _rDistribute(LP_WALLET, lpShare);
    _rDistribute(GENESIS, rDistribution - lpShare);

    emit ReserveDistributed();
}
```

**Recommendation:** Convert `burnShare` to `tValue` when passing.

**Resolution:** Fixed

## 5.2. Low/Info severity

### 5.2.1. Lows, Informational issues and code suggestions

**Description:**

**1.** in `reflect` use `reflectionFromToken` to convert amount to `rAmount`.

2. in `distributeReserve` add `minDistributeAmount` to prevent roundings.

3. in `_distributeTokens` there's a typo: `x28LiquidyPool` should be `x28LiquidITyPool`

4. in `_swapEthToX28` approval is not cleared after transferring the surplus weth, on Base transactions are cheaper.

5. in `BnB lastBuyBurn` is not set in the constructor. It can be set to the presale finalization timestamp.

6. in `excludeAccountFromReflections()` remove the `_excluded.length == 22` check.

**Resolution:** Fixed - 1,3,4,5,6