SB SECURITY

Janus PLS

Security Review

Oct 27, 2024

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	Janus PLS
Repository	Private
Commit Hash	92a6dfe2c9b5d9f16a8ead33add2741476fa4d00
Resolution	e918d9f41c071b3b340e449e9a1d1a441b36a59c
Timeline	Audit: October 27

Scope

DevDistribute.sol
JNSInvestmentPool.sol
WonderlandTreasury.sol

Issues Found

Critical Risk	1
High Risk	1
Medium Risk	0
Low/Info Risk	3
Governance Risk	1

5. Findings

5.1. Critical severity

5.1.1. No PLS token can be received

Severity: Low Risk

Description: `JNSInvestmentPool` is designed to receive `PLS` investment shares but since `Janus` uses `transfer` instead of `call` all the native transfers will revert with OOG exception in the `receive` functions because they're trying to wrap the tokens to `WPLS`.

```
receive() external payable {
    if (msg.sender != WPLS) IWPLS(WPLS).deposit{value: msg.value}();
}
```

Recommendation:

1. Remove the wrapping from the receive functions.

2. Modify the contracts to wrap the `PLS` tokens in the distribute functions.

Resolution: Fixed

5.2. High severity

5.2.1. `lastDistribution` is not updated

Severity: High Risk

Description: When calling `JNSInvestmentPool.distributePLS()`, `lastDistribution` is not updated to `block.timestamp`, which will allow the function to be spammed as soon as there are WPLS tokens inside, bypassing the cooldown check.

```
function distributePLS() external nonReentrant {
    if (!isWhitelisted(msg.sender)) revert Unauthorized();
    if (block.timestamp < lastDistribution + interval) revert Cooldown();
    IERC20 wpls = IERC20(WPLS);
    uint256 balance = wpls.balanceOf(address(this));
    if (balance == 0) revert InsufficientBalance();
    uint256 distributionAmount = capPerDistribution < balance ? capPerDistribution : balance;
    distributionAmount = _processIncentiveFee(wpls, distributionAmount);
    uint256 buyBurnAmount = distributionAmount * buyBurnShareBPS / 10000;
    wpls.safeTransfer(JNSBuyAndBurn, buyBurnAmount);
    _swapTokens(distributionAmount - buyBurnAmount);
}
```

Recommendation: Set `lastDistribution` to `block.timestamp` at end of `distributePLS()` and also make sure to assign start time in the constructor.

Resolution: Fixed

5.3. Low/Info severity

5.3.1. Superfluous check in constructors

Severity: Low Risk

Description: In the `JNSInvestmentPool` and `WonderlandTreasury` constructors, there is a check for `owner_ == address(0)`. But this check is al ready present in the Ownable constructor.

Recommendation: Remove the `owner_ == address(0)` check.

Resolution: Fixed

5.3.2. No deadline and slippage specified

Severity: Low Risk

Description: `distributePLS` does swap from `PLS` to `BUNS` but the caller has no way to specify the deadline as it is hardcoded.

As a result, swaps can happen in unfavorable conditions for the protocol. Regarding sandwich attacks, `capPerDistribution` serves as a protection and also due to the fees of BUNS attack won't be profitable at all.

Recommendation: Allow the caller to specify a custom `deadline`.

Resolution: Fixed

5.3.3. No minimum amount for distributePLS

Severity: Informational Risk

Description: There is no minimum amount for both `distributePLS()` functions. Therefore, if the amount is small enough, the incentive fee calculation will not be 0.1 and will result in 0.

Recommendation: Consider adding a minimum amount of WPLS that must be in the contract to call `distributePLS()`.

Resolution: Acknowledged

5.4. Governance risks

5.4.1. Owner can steal tokens from Wonderland

Description: Owner can withdraw `WPLS` tokens from the `WonderlandTreasury` contract.

```
function recoverToken(address tokenAddress) external onlyOwner {
    IERC20 token = IERC20(tokenAddress);
    uint256 balance = token.balanceOf(address(this));
    if (balance == 0) revert ZeroBalance();
    token.safeTransfer(msg.sender, balance);
}
```

Resolution: Acknowledged