SB SECURITY

Voyage

Security Review

Nov 7, 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	Voyage
Repository	Private
Commit Hash	9ff5a715d1e179032216a85d7bd304235b55f5c3
Resolution	4e741c31645ba4b7b0024f069a144e53d980d8d7
Timeline	October 21 - October 24, 2024

Scope

VoyageFeeHandler.sol
VoyageStaking.sol
OperatingSystemStaking.sol
VoyageLiquidityMining.sol
VoyageTokenStaking.sol
OperatingSystem.sol
Voyage.sol
VoyageVesting.sol

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	6
Low/Info Risk	6

5. Findings

5.1. Medium severity

5.1.1. Voyage is missing slippage and proper deadline check

Severity: Medium Risk

Description:

- `Voyage::_swapSellTax`

- `Voyage::_swapBuyTax`

- `VoyageFeeHandler::_swapUSDTToVoyage`

All 3 functions are hardcoding `0` as a slippage and using `block.timestamp` as a deadline.

As a result, all of them will be exposed to price manipulations. Functions in the Voyage token are callable from anyone and callers can sandwich themselves taking profit from the tax swaps. Being able to set the `swapThreshold` will partially lower the impact and the profit of the potential attacker.

On the other hand, using `block.timestamp` as a deadline will disable staleness checks of the `UniswapRouter` because when the swaps are executed we will **always** have `deadline == block.timestamp`.

Recommendation: In the functions where `amountOutMin` is hardcoded to 0, replace it with the appropriate amounts that will prevent sandwich attacks. Also, where `block.timestamp` is used as a deadline make sure to allow users to pass time as an argument, using the current timestamp as a base and adding seconds to it also won't work because it always takes it at the time of execution.

Resolution: Acknowledged

5.1.2. Voyage token can't be paused

Severity: Medium Risk

Description: `Voyage` token is intended to be pausable, but doesn't expose public `pause/unpause` functions. As a result the `paused` check in the `_beforeTokenTransfer` is not used and there will be no way to pause the token transfers.

Here is the NATSPEC of the `ERC20Pausable` contract:

```
IMPORTANT: This contract does not include public pause and unpause functions. In
 * addition to inheriting this contract, you must define both functions, invoking the
 * {Pausable-_pause} and {Pausable-_unpause} internal functions, with appropriate
 * access control, e.g. using {AccessControl} or {Ownable}. Not doing so will
 * make the contract unpausable.
```

Recommendation: Create external functions to pause and unpause the token with applied `onlyOwner` modifier.

Resolution: Fixed

5.1.3. Reserves are not synced before calculating sell amount in swapTax

Severity: Medium Risk

Description: In `Voyage::_swapSellTax` there is a risk reserves not to represent the actual tokens in the pair. This is possible because anyone can donate either one of the tokens to the pair with **direct** transfer front-running `_swapSellTax`, making the `balance > reserves`.

This is problematic because `sellAmount` heavily relies on reserves being up to date, otherwise the value will be based on stale values and when a swap happens, **donations** will also be included in the swapped amounts and the ETH that will be received will be different.

Recommendation: pair.sync() must be called before fetching the reserves, which will perform an update to the pair and will make the reserves and balances equal, guaranteeing correct `sellAmount`.

Resolution: Fixed

5.1.4. _swapUSDTToVoyage must use swapExactTokensForTokensSupportingFeeOnTransferTokens

Severity: Medium Risk

Description: _swapUSDTToVoyage is responsible from buying Voyage tokens with USDT, but since Voyage has fees on transfers UniswapV2Router::swapExactTokensForTokensSupportingFeeOnTransferTokens must be used. Otherwise, Oracle, calling the processFee function won't be able to estimate how much _minVoyageAmount to provide as an argument. The fluctuations of the price + the fees taken will force the Chainlink Automation Oracle to provide less minVoyageAmount, otherwise function will be reverting and the LINK tokens used will be wasted.

```
function _swapUSDTToVoyage(uint _amount, uint _minVoyageAmount) internal {
    IERC20(usdt).safeIncreaseAllowance(router, _amount);

    address[] memory path = new address[](2);
    path[0] = usdt;
    path[1] = voyage;
    IUniswapV2Router02(router).swapExactTokensForTokens(
        _amount,
        _minVoyageAmount,
        path,
        address(this),
        block.timestamp + 100
    );
}
```

Recommendation: Consider using the router function which takes care of the fees by performing the slippage check on the net value.

Resolution: Fixed

5.1.5. `recoverERC20()` should add ETH case

Severity: Medium Risk

Description: `recoverERC20` cannot sweep ETH, but in reality, it will be needed because of how the liquidity is provided to `Uniswap` pairs.

`Voyage::_swapSellTax` will be accumulating constant `Voy` and `Eth` dust while trying to provide liquidity to the pair, the problem is that we don't know the exact amounts of both tokens that will be supplied and if their rate is the same as the one in the pool, but the `UniswapRouter` will and will take only that much of tokens keep the ratio the same.

Although the entire contract balance is forwarded to the router, the leftovers are refunded to the caller.

Recommendation: Since the `_swapSellTax` is working as expected, by first swapping the `Voyage` token for `ETH` and the supplying both tokens we only have to add a `ETH` case in the recover function.

Resolution: Fixed

5.1.6. `VoyageTokenStaking` always uses `Voyage` as both staking and reward token

Severity: Medium Risk

Description: `VoyageTokenStaking` assign `Voyage` as both staking and reward tokens:

```
constructor(
    address _voyage,
    address _operatingSystem,
    uint _totalRewards,
    uint _minimumDeposit,
    uint _apr1Month,
    uint _apr3Month,
    uint _apr6Month,
    uint _apr12Month
)
    VoyageStaking(
        _voyage,
        _voyage,
        _totalRewards,
        _minimumDeposit,
        _apr1Month,
        _apr3Month,
        _apr6Month,
        _apr12Month
    )
{
    require(
        _operatingSystem != address(0),
        'VoyageTokenStaking: _operatingSystem cannot be the zero address'
    );
    receipt = OperatingSystem(_operatingSystem);
}
```

Despite the intentions of the protocol team to allow users to stake Voyage tokens in exchange for other reward tokens. As a result, if not anticipated at deployment this won't be possible.

Recommendation: Add rewardToken as a `VoyageTokenStaking` constructor argument and pass it to the `VoyageStaking`.

Resolution: Fixed

5.2. Low/Info severity

5.2.1. No way to mint new Voyage tokens and supply will be only initial supply

Severity: Low Risk

Description: Voyage token has a capped supply of 100 million tokens and therefore all of them are minted to the deployer. The governance issue is that he holds the entire supply and can do whatever he wants with the tokens.

Recommendation: Consider distributing the tokens, according to the allocation mentioned in the docs.

Resolution: Acknowledged

5.2.2. setTaxFee is missing 100% check

Severity: Low Risk

Description: Owner can set the buy + sell fees to be greater than 100%, which will block the token transfers for all the contracts that are not excluded. This is because there is no check in the `setTaxFee` function, capping the max % that will be taken as a fee.

```
function setTaxFee(
    uint256 liquidityFee,
    uint256 marketingFee
) external onlyOwner {
    taxInfo.liquidityFee = liquidityFee;
    taxInfo.marketingFee = marketingFee;

    emit TaxFee(liquidityFee, marketingFee);
}
```

Recommendation: Add a check preventing the setter from changing the fees to or above 100%, where `maxFeePossible` can be immutable variable.

Resolution: Fixed

5.2.3. Missing event emissions

Severity: Low Risk

Description: `setMaxHoldingPerWallet, blacklist, includeFromMaxHoldingCheck, excludeFromMaxHoldingCheck` - all these functions are not emitting events and won't be able to notify the off-chain systems that are interested in these actions.

Recommendation: Consider creating new events and emitting them from the respective functions.

Resolution: Acknowledged

5.2.4. setMaxHoldingPerWallet should have 100% check

Severity: Low Risk

Description: `setMaxHoldingPerWallet` is missing a check which to prevent users from being allowed to hold up-to 100% of the total supply.

Recommendation: Add the following check in the setter function, where `maxAllowedHoldings` can be a immutable variable.

Resolution: Acknowledged

5.2.5. Variable shadowing

Severity: Low Risk

Description: In `Voyage` token, `owner` variable is being shadowed in the `transfer` function, even though there are no security implications, it will confuse the readers.

Recommendation: Use directly `msgSender` as an argument to the `_transferWithTax` function.

Resolution: Fixed

5.2.6. Informational notes

Severity: Informational Risk

Description:

- make sure to exclude all the contracts in scope that will be using `Voyage` from the buy and sell fees, otherwise, their accounting will be broken.
- anyone can impersonate himself as a `Uniswap` pair and force the senders to pay buy/sell taxes, he only needs to expose public `token0` and `token1`, one of them returning the `Voyage` token
- if an excluded address buys Voyage from the Uni pair and the swap threshold is achieved his transaction will be reverting due to the `lock` modifier in the UniswapV2Pair contract's functions. He should perform a transfer to someone else first in order to swap the buy tax first.

Resolution: Acknowledged