PRISMA SHIELD

SPACEBOTS

DEEP LOGIC AUDIT REPORT

SpaceBOTS

DEC 27 2022

PRISMA SHIELD

Table of Contents

Overview

This audit is for the SpaceBOTS contracts. SpaceBotStation is the main contract used to create new SpaceBot contracts and where users interact to stake and unstake their tokens. SpaceBot is the contract that stores the user funds and is used by the team to swap between that contract's token pair.

Project Summary

Project Name	SpaceBOTS
Network	Multiple Networks
Language	Solidity

What is a Deep Logic Audit?

A deep logic smart contract audit is a human-driven code review that checks all of the code business logic for bugs, mathematical errors, and security risks. The audit verifies that the code honors the whitepaper. In addition, this service includes mainnet testing and proactive communication with the project owners to ensure full comprehension of the project to provide the best possible code review quality.

Overview

Findings Summary

		Findings Resolved
Total Findings	**10**	10
High Security Fingings	3	3
Medium Security Fingings	0	0
High Logical Findings	3	3
Medium Logical Findings	1	1
Informational Findings	3	3

ID	Section	Type	Severity	Page	Status
SBS-01	SpaceBotStation	Logical	High	7-8	Resolved
SBS-02	SpaceBotStation	Logical	High	9	Resolved
SBS-03	SpaceBotStation	Logical	Medium	10	Resolved
SBS-04	SpaceBotStation	Logical	Informational	11	Resolved
SBS-05	SpaceBotStation	Logical	Informational	12	Resolved
SBS-06	SpaceBotStation	Logical	Informational	13	Resolved
SBT-01	SpaceBot	Security	High	16-17	Resolved

Overview

Findings Summary

ID	Section	Type	Severity	Page	Status
SBT-02	SpaceBot	Security	High	18-19	Resolved
SBT-03	SpaceBot	Security	High	20	Resolved
SBT-04	SpaceBot	Logical	High	21	Resolved

Contract Addresses

Caution: this audit does not ensure that the team performs profitable trades. It only ensures that user funds are safe in the vaults and can be withdrawn correctly.

Moreover, the contract allows interaction with any arbitrary contract (controlled by the contract owner). This can present a security risk to user funds if the interaction is with a malicious or buggy contract.

Please ensure trust in the team before interacting with this contract.

Please ensure you are interacting with the correct contract addresses.

SpaceBotStation (BSC)
0x3A405325B2D0f18779CbfBEa007eE036a2990D98

Please ensure the following in the contract:

- Open https://bscscan.com/address/0x3A405325B2D0f18779CbfBEa007eE036a2990D98#readContract
- Ensure that the owner field is 0xc2723929f3bd8a95ee7fb5ff2ada6eae94215f8b

SpaceBotStation (Polygon)
0x3A405325B2D0f18779CbfBEa007eE036a2990D98

Please ensure the following in the contract:

- Open https://polygonscan.com/address/0x3A405325B2D0f18779CbfBEa007eE036a2990D98#readContract
- Ensure that the owner field is 0xe8f92dce43b468abaf940079ea35433e7d891b3a

SpaceBotStation (Avalanche)
0x3A405325B2D0f18779CbfBEa007eE036a2990D98

Please ensure the following in the contract:

- Open https://snowtrace.io/address/0x3A405325B2D0f18779CbfBEa007eE036a2990D98#readContract
- Ensure that the owner field is 0xfdab0f0c006fd12579600c08b47510ecc883c7a6

Audit Findings

SpaceBotStation

SBS-01 - Logical High Severity

The stakeInVault and unstakeFromVault functions have multiple issues that would prevent them from functioning correctly and would result in a negative effect to user funds:

- They do not correctly calculate users' shares of the deposited funds, as they do not properly account for the changes in the ratio of the tokens when they're traded
- They loop over externally manipulated arrays and mappings, which could result in the functions being uncallable if those data structures grow too large to cause the gas required to call them to exceed the block gas limit
- The unstakeFromVault function can be disabled by the contract owner, preventing users from withdrawing their funds

Recommendation

Change the logic of the functions to work similarly to how user shares are calculated when adding and removing liquidity in DEX liquidity pools. The logic involves assigning each user a share amount that is proportional to the amount of one of the tokens in the vault. Users can only stake and unstake tokens in the same ratio as that of the tokens in the vault in the same transaction. Moreover, always allow users to withdraw their funds.

The code change implementing this recommendation is supplied separately alongside this audit report. Three view functions were added as part of the recommendation:

- desiredAmount, which given the amount of one token belonging to a vault, returns the amount of the other token in the same ratio as that of the tokens in the vault
- userTokenAmounts, which returns the amount of tokens a user owns in a vault
- shareholders, which returns the list of all users who own tokens in a vault

Audit Findings

SpaceBotStation

SBS-01 - Logical High Severity

Resolution
The team has accepted the recommended code change.

The stakeInVault and unstakeFromVault functions implement similar logic as that of adding and removing liquidity in DEX liquidity pools, ensuring that users always have access to the correct amount of tokens.

Audit Findings

SpaceBotStation

SBS-02 - Logical High Severity

The zapInVault function has multiple errors that would result in it not working correctly:

- It does not transfer the user tokens to the contract before calling the swapExactTokensForTokens router function
- It assumes that the tokens should be split 50/50, and does not take into account the ratio of tokens in the vault
- It calls the stakeInVault function as an external function call, which would result in the shares being assigned to the SpaceBotStation contract instead of to the user who is staking the tokens

Recommendation

Transfer the tokens being zapped from the user to the contract. Calculate how much should be swapped to the other tokens using the ratio of tokens in the vault and price of the tokens. Call stakeInVault as an internal function call (rather than external) by changing it to public and calling it without using this. Then transfer the excess funds that were not staked back to the user.

The math for the correct calculation to be used for the zap has been supplied alongside this audit report.

Resolution

The team has decided to remove the zapInVault function.

The team has decided to implement the zap functionality directly in their dApp. They have been advised by Prisma Shield on the correct way to do that, but because the functionality is outside of the audited smart contract, it is not audited by Prisma Shield.

Audit Findings

SpaceBotStation

SBS-03 - Logical Medium Severity

The addVault function requires inputing the decimals of each token as a function parameter. This increases the likelihood of a human error occurring in the function call, which could potentially result in a faulty vault being created.

Recommendation

Do not supply the token decimals to the function, and instead use the decimals function of the token smart contract directly when required.

The code change implementing this recommendation is supplied separately alongside this audit report.

Resolution

The team has accepted the recommended code change.

Whenever token decimals are required, the decimals function of the token smart contract is used.

Audit Findings

SpaceBotStation

SBS-04 - Logical Informational Severity

The addTokenForVault function and associated ERC20TokenInfo and ercTokenCount state variables don't have any useful functionality.

Recommendation

Remove the addTokenForVault function and the ERC20TokenInfo and ercTokenCount state variables.

Resolution

The team has resolved this issue.

The addTokenForVault function and the ERC20TokenInfo and ercTokenCount state variables have been removed.

Audit Findings

SpaceBotStation

SBS-05 - Logical Informational Severity

The getVaults function is redundant because the automatically generated vaultInfo and vaultCount functions already provide the same information.

Recommendation

Remove the getVaults function.

Resolution

The team has resolved this issue.

The getVaults function has been removed.

Audit Findings

SpaceBotStation

SBS-06 - Logical Informational Severity

The _safeTransferFrom function and the IERC20.sol, Ownable.sol, ReentrancyGuard.sol imports can all be replaced with the corresponding community-audited OpenZeppelin files that are the current industry standard.

Recommendation
Use the necessary OpenZeppelin imports.

The code change implementing this recommendation is supplied separately alongside this audit report.

Resolution
The team has accepted the recommended code change.

The necessary OpenZeppelin files are being imported and used, and the _safeTransferFrom function and the custom IERC20.sol, Ownable.sol, ReentrancyGuard.sol imports have all been removed.

Overview

SpaceBotStation

- This contract allows users to stake and unstake pairs of tokens in vaults, that can be used by the team to perform automated trades using different trading strategies on behalf of the users.

- The adminDelegator publicly viewable variable contains a regular non-multisig address that can be used to call the addVault function. The contract owner has admin powers to change this value.

- The contractEnabled publicly viewable variable contains a boolean, where if true, users are allowed to stake their, and if false, users are not allowed to stake. The contract owner has admin powers to change this value.

- The vaultInfo publicly viewable variable contains the information about all the available vaults (SpaceBot contracts). The vaultCount publicly viewable variable contains the total number of available vaults. The adminDelegator has admin powers to change these values.

- The dexAggs publicly viewable variable contains the list of contracts that can be used to trade the tokens in the vaults. The dexAggsLen publicly viewable variable contains the number of contracts. The contract owner has admin powers to change this value.

- The totalVaultShares publicly viewable variable contains the total number of shares per vault. The userShares publicly viewable variable contains the amount of shares that each user owns per vault. The shareholders publicly viewable functions returns the list of all shareholders in the specified vault.

- The stakeInVault function allows users to stake their tokens in the specified vault. The contract owner has admin powers to disable this function from being called.

Overview

SpaceBotStation

- The unstakeFromVault function allows users to unstake their tokens from the specified vault.

- The desiredAmount publicly viewable function provides the corresponding amount of tokens relative to other based on the ratio of the two tokens in the specified vault. This allows anyone to easily find out how much they should stake of one token specifiying the amount of the other token and the desired vault.

- The userTokenAmounts publicly viewable function returns the amount of tokens the specified user owns in the specified vault.

Audit Findings

SpaceBot

SBT-01 - Security High Severity

The swapTrade function allows the interaction with any arbitrary contract (gated by the contract owner), which puts user funds at risk. It does not guarantee that the correct amount of tokens is being traded, and does not ensure that only the two tokens of the vault are being traded. It fully relies on the correctness of the data passed to the function by the adminDelegator, which is a regular (non-multisig) address. It also allows the token balance to go down to zero, which breaks the staking and unstaking logic.

Recommendation

Change the function signature to receive arguments about the token that this being swapped out and its amount (instead of the token being swapped in). Perform the approval of that exact amount in the swapTrade function. Add logic to ensure that exact amount specified was swapped out, not allowing to bring the vault balance down to zero, and check that the amount received of the other token is close to what a popular DEX in that chain (e.g. PancakeSwap for BSC) would have given for that trade. The trading fees should be taken from the actual amount of the token received.

The code change implementing this recommendation is supplied separately alongside this audit report. It includes changes to the constructor, and the following two functions were added as part of this recommendation:

- setPath, which is used to set the path used to get the approximate amount of tokens that should be received from the trade (the trade itself can go through a different DEX with a different price)
- setSlippage, which is used to set the allowed percentage to be received of the token less than approximate amount. This percentage is between 0% and 1% (i.e. the actual amount of tokens received from the trade can only be up to 1% less than the approximate amount)

Audit Findings

SpaceBot

SBT-01 - Security High Severity

Resolution

The team has accepted the recommended code change.

The swapTrade function has applied the necessary logic to ensure the three most important security requirements in that contract:

- It checks (but doesn't guarantee) that only the two tokens of the vault are being traded (i.e. no other tokens outside of the two tokens of the vault are being traded). It cannot be easily guaranteed due to the function allowing the trade to happen using any arbitrary function call with any arbitrary data through any arbitrary DEX or DEX aggregator
- It checks (but doesn't guarantee) that a reasonable amount of the tokens is being received. It cannot be easily guaranteed due to the function allowing the trade to happen through any arbitrary DEX or DEX aggregator, which could result in different prices from the DEX that is used to get the approximate price
- Because it allows interaction with any arbitrary contract, it only approves the amount to be traded in that transaction

Audit Findings

SpaceBot

SBT-02 - Security High Severity

The addDexAgg function can be called by the adminDelegator (which is a regular non-multisig address) to allow interacting with any arbitrary contract, and it approves the added address to spend an unlimited amount of the tokens in the vault. This opens a massive security vulnerability, where if a malicious address was added in that function, it can steal all of the user funds.

Recommendation

Change the function to be onlyOwner, where the contract owner is the team's multisig address, and remove the approval code from that function. In addition, move the function and corresponding dexAggs variable to the SpaceBotStation contract to make it easier to add new DEXes and DEX aggregators that are useable by all vaults in a single function call rather than doing it in each vault separately. Moreover, add a removeDexAgg function which can be used to remove DEXes and DEX aggregators if necessary:

```
contract SpaceBotStation {
    ...
    address[] public dexAggs;
    uint256 public dexAggsLen;

    function addDexAgg(address aggAddress) external onlyOwner {
        dexAggs.push(aggAddress);
        dexAggsLen++;
    }
    function removeDexAgg(uint256 aggSelector) external onlyOwner {
        dexAggs[aggSelector] = dexAggs[dexAggs.length - 1];
        dexAggs.pop();
        dexAggsLen--;
    }
    ...
}
```

Audit Findings

SpaceBot

SBT-02 - Security High Severity

Resolution
The team has resolved this issue.

The addDexAgg function has been changed to onlyOwner, it no longer performs the token approvals, and it has been moved to the SpaceBotStation contract. The removeDexAgg function has been added.

Audit Findings

SpaceBot

SBT-03 - Security High Severity

The updateFees function does not enforce an upper limit for the percentage of tokens that is transferred to the teamWalletAddr from trades. If set incorrectly, it can result in more tokens being transferred than intended.

Recommendation

Add a require statement in the updateFees function to enforce a reasonable upper limit for that percentage:

```
function updateFees(
    uint256 _newFees,
    address _newFeesRecipient
) external onlyOwner {
    require(
        _newFees <= 4000000000000000,
        "Exceeded maximum allowed fee of 0.4%"
    );
    fees = _newFees;
    teamWalletAddr = _newFeesRecipient;
}
```

Resolution

The team has resolved this issue.

A require statement has been added to only allow the percentage to be set to a maximum of 0.4%.

Audit Findings

SpaceBot

SBT-04 - Logical High Severity

The SpaceBot contract does not approve the SpaceBotStation contract to withdraw the tokens. This implies that users would not be able to unstake their tokens.

Recommendation

Add approve statements allowing the SpaceBotStation contract to withdraw the tokens:

```
constructor(address token1, address token2, SpaceBotStation station) {
    TOKEN_1 = IERC20(token1);
    TOKEN_2 = IERC20(token2);
    TOKEN_1.safeApprove(address(STATION), type(uint256).max);
    TOKEN_2.safeApprove(address(STATION), type(uint256).max);
    ...
}
```

Resolution

The team has resolved this issue.

The SpaceBot contract approves the SpaceBotStation contract to withdraw the tokens.

Overview

SpaceBot

- This contract holds the user funds, and allows the team to trade between the tokens and collect fees from each trade.

- The ROUTER publicly viewable variable contains the address of the DEX that is used to get the approximate amount of tokens received. On BSC, this is the PancakeSwap router address.

- The TOKEN_1 and TOKEN_2 publicly viewable variables contain the addresses of the two tokens belonging to that vault.

- The STATION publicly viewable variable contains the address of the SpaceBotStation contract.

- The slippage publicly viewable variable contains a percentage between 99% and 100%. It defaults to 99.7%. The amount of tokens received in the swapTrade function can be that percentage of the approximate amount calculated from the ROUTER contract. In other words, the actual amount of tokens received has to be at least 99% of the approximate amount. The contract owner has admin powers to change this value.

- The adminDelegator publicly viewable variable contains a regular non-multisig address that can be used to call the swapTrade function. The contract owner has admin powers to change this value.

- The teamWalletAddr publicly viewable variable contains the address that receives the fees from the trades in the swapTrade function. The contract owner has admin powers to change this value.

Overview

SpaceBot

- The fees publicly viewable variable contains the percentage that is taken as fees from the trades in the swapTrade function. It can be between 0% and 0.4%, and defaults to 0.2%. The fee percentage is taken from thetokens received in a trade. The contract owner has admin powers to change this value.

- The path1 and path2 publicly viewable variables contain the paths that are used in ROUTER to get the approximate amount received in each trade. path2 is always the reverse of path1. In other words, path1 is used to get the amount when swapping from TOKEN_1 to TOKEN_2, and path2 is used to get the amount when swapping from TOKEN_2 to TOKEN_1. The pathLen publicly viewable variable contains the length of those arrays. The contract owner has admin powers to change these values.

- The swapTrade function is used by the team to swap between TOKEN_1 and TOKEN_2. The adminDelegator has admin powers to call this function.

How to Interpret Findings

Security - High Severity

Indicates that users' funds are at risk or that there is a high probability of exploitation.

Security - Medium Severity

No risk to the protocol or those who interact with it, however it should be highlighted nonetheless.

Logical - High Severity

Indicates that the errors puts users' funds at risk, or can result in significant functional failure in the code.

Logical - Medium Severity

Indicates some functional failure or discrepancy in the code.

Logical - Informational

Minor discrepancy between the intended functionality of the code and the implementation, which does not result in functional failure, or a recommendation to improve the logic.

Yellow Text

Indicates centralization of control and admin powers.

Disclaimer

The information in this deep logic audit report objectively describes the smart contracts being audited, and points out logical and mathematical errors, security risks and vulnerabilities, and optimization opportunities in the audited code. This deep logic audit does not ensure the correctness or authenticity of any software or dApp that interacts with or claims to interact with any smart contract.

This audit report does not constitute any advice whatsoever. You are solely responsible for conducting your own due diligence and consulting your financial advisor before making any investment decisions. While our deep logic audits raise the level of security, reliability, mathematical accuracy, and logical soundness of the smart contracts reviewed, they do not amount to any form of warranty or guarantee that the reviewed smart contracts are void of any weaknesses, vulnerabilities, or bugs. Prisma Shield and its founders, employees, owners, and associates are not liable for any damage or loss of funds. Please ensure trust in the team prior to investing as this deep logic audit does not guarantee the promised use of your funds.

You can find the full disclaimer, terms and conditions, and privacy policy on the prismashield.com website.

PRISMA SHIELD

Introducing Deep Logic Smart Contract Auditing to Web3

prismashield.com

prismashield@gmail.com

PrismaShield

PrismaShield