PRISMA SHIELD

WAND INVESTMENTS

DEEP LOGIC AUDIT REPORT

WAND Main Token Exchange Contract

AUG 30 2022

PRISMA SHIELD

Table of Contents

Overview

This audit is for the main token exchange contract of WAND Investments where users can buy and sell Scepter and Baton tokens, and burn Scepter tokens for Baton tokens.

Project Summary

Project Name	WAND Investments
Network	Binance Smart Chain
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	**15**	15
High Security Fingings	1	1
Medium Security Fingings	0	0
High Logical Findings	9	9
Medium Logical Findings	2	2
Informational Findings	3	3

ID	Section	Type	Severity	Page	Status
WND-01	WandInvestments	Security	High	7	Resolved
WND-02	WandInvestments	Logical	High	8-10	Resolved
WND-03	WandInvestments	Logical	High	11-13	Resolved
WND-04	WandInvestments	Logical	High	14-15	Resolved
WND-05	WandInvestments	Logical	High	16-18	Resolved
WND-06	WandInvestments	Logical	High	19	Resolved
WND-07	WandInvestments	Logical	High	20-22	Resolved

Overview

Findings Summary

ID	Section	Type	Severity	Page	Status
WND-08	WandInvestments	Logical	High	23-24	Resolved
WND-09	WandInvestments	Logical	High	25	Resolved
WND-10	WandInvestments	Logical	High	26	Resolved
WND-11	WandInvestments	Logical	Medium	27-28	Resolved
WND-12	WandInvestments	Logical	Informational	29	Resolved
WND-13	WandInvestments	Logical	Informational	30-31	Resolved
MATH-01	mathFuncs	Logical	Medium	36	Resolved
MATH-02	mathFuncs	Logical	Informational	37	Resolved

Contract Addresses

Please ensure you are interacting with the correct contract addresses.

WandInvestments
0x62Ca5Ca1F7A77711B1ab543E3Fab40bD5fEEd6E6

Audit Findings

WandInvestments

WND-01 - Security High Severity

The multisig gnosis address can't be used directly to deploy the contract on the blockchain, and therefore is not the contract owner on deployment. The deploying non-multisig address will need to explicitly call the transferOwnership function post-deployment. If ownership is not transferred to the multisig address, the non-multisig address will have unilateral control over the contract, and will be able to call all admin functions unilaterally.

Recommendation

Call the _transferOwnership function with the multisig address as the argument in the constructor to ensure that the multisig address is the contract owner from deployment:

```
constructor() {
    // Multisig address is the contract owner.
    _transferOwnership(address(0) /*TODO: Change to the multisig address*/);
}
```

Resolution

The team has resolved this issue.

The multisig address is the contract owner on deployment.

Audit Findings

WandInvestments

WND-02 - Logical High Severity

The semantics of the function getCircSupplyXDays does not correctly match the intention. This function is supposed to return what the Scepter token circulating supply was X days ago. Instead, it is returning the amount of Scepter tokens bought minus the amount sold and burned over the previous X days. This will result in incorrect math when calculating the sptrGrowthFactor and sptrSellFactor, as well as some transactions failing when the variable circulatingSupplyXDays is being subtracted from to result in a negative value (it is an unsigned integer, always supposed to be non-negative).

In addition, it requires buy and sell functions to be called everyday to update. If a single day passes without any trading volume, the circulatingSupplyXDays state variable will not be updated, which will result in an incorrect calculation.

Audit Findings

WandInvestments

WND-02 - Logical High Severity

Recommendation

Change the code to the following:

```
mapping(uint256 => bool) private setCircSupplyToPreviousDay;

function setCirculatingSupplyXDaysToPrevious(uint256 dInArray) private returns (uint256) {
    if (setCircSupplyToPreviousDay[dInArray]) {
        return circulatingSupplyXDays[dInArray];
    }
    setCircSupplyToPreviousDay[dInArray] = true;
    circulatingSupplyXDays[dInArray] = setCirculatingSupplyXDaysToPrevious(dInArray - 1);
    return circulatingSupplyXDays[dInArray];
}

function cashOutScepter(
    uint256 amountSPTRtoSell,
    uint256 daysChosenLocked,
    string calldata stableChosen
)
    external nonReentrant
{
    ...
    setCirculatingSupplyXDaysToPrevious(dInArray);
    circulatingSupplyXDays[dInArray] -= amountSPTRtoSell;
    ...
}

function transformScepterToBaton(uint256 amountSPTRtoSwap) external nonReentrant {
    ...
    setCirculatingSupplyXDaysToPrevious(dInArray);
    circulatingSupplyXDays[dInArray] -= amountSPTRtoSwap;
    ...
}
```

Audit Findings

WandInvestments

WND-02 - Logical High Severity

Code continued...

```
function buyScepter(uint256 amountSPTRtoBuy, string calldata stableChosen) external nonReentrant {
    ...
    setCirculatingSupplyXDaysToPrevious(dInArray);
    circulatingSupplyXDays[dInArray] += amountSPTRtoBuy;
    ...
}

function wlBuyScepter(uint256 amountSPTRtoBuy, string calldata stableChosen) external nonReentrant {
    ...
    setCirculatingSupplyXDaysToPrevious(dInArray);
    circulatingSupplyXDays[dInArray] += amountSPTRtoBuy;
    ...
}

function getCircSupplyXDays() public view returns (uint256) {
    if (timeLaunched == 0) return SEED_AMOUNT;
    uint256 daySinceLaunched = (block.timestamp - timeLaunched) / SECONDS_IN_A_DAY;
    uint256 numdays = daysInCalculation / SECONDS_IN_A_DAY;
    if (daySinceLaunched < numdays) {
        return SEED_AMOUNT;
    }
    for (uint d = daySinceLaunched - numdays; d > 0; d--) {
        if (setCircSupplyToPreviousDay[d]) {
            return circulatingSupplyXDays[d];
        }
    }
    return circulatingSupplyXDays[0];
}
```

Resolution

The team has resolved this issue.

The recommended code change has been applied.

Audit Findings

WandInvestments

WND-03 - Logical High Severity

The cashOutScepter function contains multiple logical errors:

- It does not require that the variable tradingEnabled to be set to true, which would incorrectly allow that function to be called to sell Scepter tokens before launch
- It does not require that the daysChosenLocked argument is less than 10. The maximum value of that argument should be 9
- It does not check the USD calculation for integer division that results in 0. Users who sell small amounts of Scepter can end up not receiving any stable coins
- The tax the user receives increases proportionally to the number of locked days from 20% (1 locked day) to 100% (9 locked days), when it should instead decrease from 80% (1 locked day) to 0% (9 locked days)
- The amount removed from the sptrTreasuryBal does not correctly correspond to the actual USD amount transferred from the treasury
- It does not split the transferred stable coins correctly between the user and the DEV_WALLET_ADDR. It should be sending 95% of the funds (after tax) to the user, and the remaining 5% (after tax) to the DEV_WALLET_ADDR
- The corresponding Wand tokens should be burned from the SCEPTER_TREASURY_ADDR and not from the WandInvestments contract

Audit Findings

WandInvestments

WND-03 - Logical High Severity

Recommendation

Change the code to the following:

```
function cashOutScepter(
    uint256 amountSPTRtoSell,
    uint256 daysChosenLocked,
    string calldata stableChosen
)
    external nonReentrant
{
    require(tradingEnabled, "Disabled");
    require(SPTR.balanceOf(msg.sender) >= amountSPTRtoSell, "You dont have that amount!");
    require(daysChosenLocked < 10, "You can only lock for a max of 9 days");

    uint256 usdAmt = mathFuncs.decMul18(
        mathFuncs.decMul18(scepterData.sptrSellPrice, amountSPTRtoSell),
        mathFuncs.decDiv18((daysChosenLocked + 1) * 10, 100)
    );

    require(usdAmt > 0, "Not enough tokens swapped");

    uint256 dInArray = (block.timestamp - timeLaunched) / SECONDS_IN_A_DAY;
    tokensSoldXDays[dInArray] += amountSPTRtoSell;
    setCirculatingSupplyXDaysToPrevious(dInArray);
    circulatingSupplyXDays[dInArray] -= amountSPTRtoSell;

    scepterData.sptrTreasuryBal -= usdAmt;

    calcSPTRData();
```

Audit Findings

WandInvestments

WND-03 - Logical High Severity

Code continued...

```
    WAND.burn(SCEPTER_TREASURY_ADDR, amountSPTRtoSell);
    SPTR.burn(msg.sender, amountSPTRtoSell);

    if (daysChosenLocked == 0) {
        require(stableERC20Info[stableChosen].contractAddress != address(0), "Unsupported stable coin");
        IERC20 tokenStable = IERC20(stableERC20Info[stableChosen].contractAddress);

        uint256 usdAmtTrf = usdAmt / 10**(18 - stableERC20Info[stableChosen].tokenDecimals);
        uint256 usdAmtToUser = mathFuncs.decMul18(usdAmtTrf, mathFuncs.decDiv18(95, 100));

        require(usdAmtToUser > 0, "Not enough tokens swapped");

        _safeTransferFrom(tokenStable, SCEPTER_TREASURY_ADDR, msg.sender, usdAmtToUser);
        _safeTransferFrom(tokenStable, SCEPTER_TREASURY_ADDR, DEV_WALLET_ADDR, usdAmtTrf - usdAmtToUser);
    } else {
        withheldWithdrawals[msg.sender].amounts = usdAmt;
        withheldWithdrawals[msg.sender].timeUnlocked =
            block.timestamp + (daysChosenLocked * SECONDS_IN_A_DAY);
    }

    timeSold[msg.sender] = block.timestamp;
    if (SPTR.balanceOf(msg.sender) == 0 && BTON.balanceOf(msg.sender) == 0) {
        initialTimeHeld[msg.sender] = 0;
    }

    emit sceptersSold(msg.sender, amountSPTRtoSell);
}
```

Resolution

The team has resolved this issue.

The recommended code change has been applied.

Audit Findings

WandInvestments

WND-04 - Logical High Severity

The cashOutBaton function contains multiple logical errors:

- It does not transfer the corresponding USD to the user
- It does not check the USD calculation for integer division that results in 0. Users who sell small amounts of Baton can end up not receiving any stable coins
- It does not update the btonTreasuryBal and the btonHoldings state variables

Recommendation

Change the code to the following:

```
function cashOutBaton(uint256 amountBTONtoSell, string calldata stableChosen) external nonReentrant {
    require(tradingEnabled, "Disabled");
    require(BTON.balanceOf(msg.sender) >= amountBTONtoSell, "You dont have that amount!");
    require(stableERC20Info[stableChosen].contractAddress != address(0), "Unsupported stable coin");

    IERC20 tokenStable = IERC20(stableERC20Info[stableChosen].contractAddress);

    uint256 usdAmt = mathFuncs.decMul18(getBTONRedeemingPrice(), amountBTONtoSell);
    uint256 usdAmtTrf = usdAmt / 10**(18 - stableERC20Info[stableChosen].tokenDecimals);

    require(usdAmtTrf > 0, "Not enough tokens swapped");

    btonTreasuryBal -= usdAmt;

    btonHoldings[msg.sender].timeInit = block.timestamp;
    btonHoldings[msg.sender].amounts -= amountBTONtoSell;

    BTON.burn(msg.sender, amountBTONtoSell);
    _safeTransferFrom(tokenStable, BATON_TREASURY_ADDR, msg.sender, usdAmtTrf);

    timeSold[msg.sender] = block.timestamp;
    if (SPTR.balanceOf(msg.sender) == 0 && BTON.balanceOf(msg.sender) == 0) {
        initialTimeHeld[msg.sender] = 0;
    }
}
```

Audit Findings

WandInvestments

WND-04 - Logical High Severity

Resolution

The team has resolved this issue.

The recommended code change has been applied.

Audit Findings

WandInvestments

WND-05 - Logical High Severity

The transformScepterToBaton function contains multiple logical errors:

- It does not check the USD calculation for integer division that results in 0. Users who transform small amounts of Scepter to Baton can end up not transferring the corresponding Usdc from the Scepter treasury to the Baton treasury
- It does not update the btonTreasuryBal state variable, which would result in incorrectly calculating the getBTONRedeemingPrice function
- The corresponding Wand tokens should be burned from the SCEPTER_TREASURY_ADDR and not from the WandInvestments contract
- The corresponding Usdc tokens should be transferred from the SCEPTER_TREASURY_ADDR and not from the WandInvestments contract

Audit Findings

WandInvestments

WND-05 - Logical High Severity

Recommendation

Change the code to the following:

```
function transformScepterToBaton(uint256 amountSPTRtoSwap) external nonReentrant {
    require(tradingEnabled, "Disabled");
    require(SPTR.balanceOf(msg.sender) >= amountSPTRtoSwap, "You dont have that amount!");

    uint256 btonTreaAmtTrf = mathFuncs.decMul18(
        mathFuncs.decMul18(scepterData.sptrBackingPrice, amountSPTRtoSwap),
        mathFuncs.decDiv18(9, 10)
    );
    uint256 toTrf = btonTreaAmtTrf / 10**12;

    require(toTrf > 0, "Not enough tokens swapped");

    IERC20 tokenStable = IERC20(stableERC20Info["USDC"].contractAddress);

    uint256 dInArray = (block.timestamp - timeLaunched) / SECONDS_IN_A_DAY;
    tokensSoldXDays[dInArray] += amountSPTRtoSwap;
    setCirculatingSupplyXDaysToPrevious(dInArray);
    circulatingSupplyXDays[dInArray] -= amountSPTRtoSwap;

    scepterData.sptrTreasuryBal -= btonTreaAmtTrf;

    calcSPTRData();

    btonTreasuryBal += btonTreaAmtTrf;

    btonHoldings[msg.sender].timeInit = block.timestamp;
    btonHoldings[msg.sender].amounts += amountSPTRtoSwap;

    WAND.burn(SCEPTER_TREASURY_ADDR, amountSPTRtoSwap);
    SPTR.burn(msg.sender, amountSPTRtoSwap);
    BTON.mint(msg.sender, amountSPTRtoSwap);
    _safeTransferFrom(tokenStable, SCEPTER_TREASURY_ADDR, BATON_TREASURY_ADDR, toTrf);
}
```

Audit Findings

WandInvestments

WND-05 - Logical High Severity

Resolution

The team has resolved this issue.

The recommended code change has been applied.

Audit Findings

WandInvestments

WND-06 - Logical High Severity

The buyScepter function can be abused to mint Scepter tokens for free without paying any stable coins. This is due to not checking the USD calculation for integer division that results in 0.

Recommendation

Add a check that the 95% of the usdAmtToPay value is larger than 0:

```
function buyScepter(uint256 amountSPTRtoBuy, string calldata stableChosen) external nonReentrant {
    ...
    uint256 usdAmtToTreasury = mathFuncs.decMul18(usdAmtToPay, mathFuncs.decDiv18(95, 100));

    require(usdAmtToTreasury > 0, "Not enough tokens swapped");

    _safeTransferFrom(tokenStable, msg.sender, SCEPTER_TREASURY_ADDR, usdAmtToTreasury);
    _safeTransferFrom(tokenStable, msg.sender, DEV_WALLET_ADDR, usdAmtToPay - usdAmtToTreasury);
    ...
}
```

Resolution

The team has resolved this issue.

The recommended code change has been applied.

Audit Findings

WandInvestments

WND-07 - Logical High Severity

The wlBuyScepter function contains multiple logical errors:

- It can be abused to mint Scepter tokens for free without paying any stable coins. This is due to not checking the USD calculation for integer division that results in 0
- It should be allowed to be used 36 hours after launch instead of 48 hours
- It does not correctly calculate the price of the Scepter tokens. 60% of the tokens purchased should be priced at a fixed 1 USD, and the remaining 40% should be priced at the sptrBuyPrice
- It should not be allowed to run for purchases below 25,000 USD
- It does not update the tokensBoughtXDays and circulatingSupplyXDays state variables, which would result in incorrect Scepter price calculations
- It should not have a maximum Scepter purchase limit
- It does not update the initialTimeHeld state variable

Audit Findings

WandInvestments

WND-07 - Logical High Severity

Recommendation

Change the code to the following:

```
function wlBuyScepter(uint256 amountSPTRtoBuy, string calldata stableChosen) external nonReentrant {
    require(tradingEnabled, "Disabled");
    require(block.timestamp > timeLaunched + 129600); // 36hrs
    require(whiteListAddresses[msg.sender], "Not Whitelisted");
    require(stableERC20Info[stableChosen].contractAddress != address(0), "Unsupported stable coin");

    IERC20 tokenStable = IERC20(stableERC20Info[stableChosen].contractAddress);

    uint256 usdAmtFixed = mathFuncs.decMul18(amountSPTRtoBuy, mathFuncs.decDiv18(60, 100));
    uint256 usdAmt =
        usdAmtFixed +
        mathFuncs.decMul18(amountSPTRtoBuy - usdAmtFixed, scepterData.sptrBuyPrice);
    uint256 usdAmtToPay = usdAmt / 10**(18 - stableERC20Info[stableChosen].tokenDecimals);

    require(usdAmt >= 25000 * DECIMALS, "Whale WL purchase has to be larger than 25K USD");
    require(tokenStable.balanceOf(msg.sender) >= usdAmtToPay, "You dont have that amount!");

    uint256 dInArray = (block.timestamp - timeLaunched) / SECONDS_IN_A_DAY;
    tokensBoughtXDays[dInArray] += amountSPTRtoBuy;
    setCirculatingSupplyXDaysToPrevious(dInArray);
    circulatingSupplyXDays[dInArray] += amountSPTRtoBuy;

    scepterData.sptrTreasuryBal += mathFuncs.decMul18(usdAmt, mathFuncs.decDiv18(95, 100));

    calcSPTRData();
```

Audit Findings

WandInvestments

WND-07 - Logical High Severity

Code continued...

```
    uint256 usdAmtToTreasury = mathFuncs.decMul18(usdAmtToPay, mathFuncs.decDiv18(95, 100));

    require(usdAmtToTreasury > 0, "Not enough tokens swapped");

    _safeTransferFrom(tokenStable, msg.sender, SCEPTER_TREASURY_ADDR, usdAmtToTreasury);
    _safeTransferFrom(tokenStable, msg.sender, DEV_WALLET_ADDR, usdAmtToPay - usdAmtToTreasury);

    SPTR.mint(msg.sender, amountSPTRtoBuy);
    WAND.mint(SCEPTER_TREASURY_ADDR, amountSPTRtoBuy);

    if (initialTimeHeld[msg.sender] == 0) {
        initialTimeHeld[msg.sender] = block.timestamp;
    }

    emit sceptersBought(msg.sender, amountSPTRtoBuy);
}
```

Resolution

The team has resolved this issue.

The recommended code change has been applied.

Audit Findings

WandInvestments

WND-08 - Logical High Severity

The claimLockedUSD function contains multiple logical errors:

- It allows anyone to claim the locked USD of anyone else by supplying any address to the _claimant argument
- It does not clear the withheldWithdrawals state variable for the user after they have successfully claimed their locked funds. This would allow users to claim any amount of funds they wish even if they don't have any locked funds
- It does not correctly calculate the stable coin amount that should be transferred to the user as it does not ensure the correct number of decimals
- It does not split the funds 95% to the user and 5% to the DEV_WALLET_ADDR

Recommendation

Change the code to the following:

```
function claimLockedUSDC(string calldata stableChosen) external nonReentrant {
    require(tradingEnabled, "Disabled");
    require(withheldWithdrawals[msg.sender].timeUnlocked != 0, "No locked funds to claim");
    require(block.timestamp >= withheldWithdrawals[msg.sender].timeUnlocked, "Not unlocked");
    require(stableERC20Info[stableChosen].contractAddress != address(0), "Unsupported stable coin");

    IERC20 tokenStable = IERC20(stableERC20Info[stableChosen].contractAddress);

    uint256 claimAmts =
        withheldWithdrawals[msg.sender].amounts /
        10**(18 - stableERC20Info[stableChosen].tokenDecimals);
    uint256 amtToUser = mathFuncs.decMul18(claimAmts, mathFuncs.decDiv18(95, 100));

    delete withheldWithdrawals[msg.sender];
    _safeTransferFrom(tokenStable, SCEPTER_TREASURY_ADDR, msg.sender, amtToUser);
    _safeTransferFrom(tokenStable, SCEPTER_TREASURY_ADDR, DEV_WALLET_ADDR, claimAmts - amtToUser);
}
```

Audit Findings

WandInvestments

WND-08 - Logical High Severity

Resolution

The team has resolved this issue.

The recommended code change has been applied.

Audit Findings

WandInvestments

WND-09 - Logical High Severity

The getBTONRedeemingPrice function does not correctly calculate half of the sptrBackingPrice, which would result in the function returning an incorrect figure.

Recommendation

Divide the sptrBackingPrice by 2 using the division operator / instead of using the mathFuncs.devDiv18 function:

```
function getBTONRedeemingPrice() public view returns (uint256) {
    uint256 btonPrice = mathFuncs.decMul18(getBTONBackingPrice(), mathFuncs.decDiv18(30, 100));
    uint256 sptrPriceHalf = scepterData.sptrBackingPrice / 2;
    if (btonPrice > sptrPriceHalf) {
        return sptrPriceHalf;
    }
    return btonPrice;
}
```

Resolution

The team has resolved this issue.

The recommended code change has been applied.

Audit Findings

WandInvestments

WND-10 - Logical High Severity

The testUpdatetimeLockedplus and testUpdatetimeLockedminus functions can be used to freely change the locked USD amounts of the user running them, and should be deleted.

Recommendation

Delete the testUpdatetimeLockedplus and testUpdatetimeLockedminus functions.

Resolution

The team has resolved this issue.

These functions have been deleted.

Audit Findings

WandInvestments

WND-11 - Logical Medium Severity

The getTokensBoughtXDays and getTokensSoldXDays functions do not account for the trading volume of the same day.

Recommendation

In the for-loops, check for <= daySinceLaunched:

```
function getTokensBoughtXDays() public view returns (uint256) {
    if (timeLaunched == 0) return tokensBoughtXDays[0];

    uint256 boughtCount = 0;
    uint d = 0;
    uint256 numdays = daysInCalculation / SECONDS_IN_A_DAY;
    uint256 daySinceLaunched = (block.timestamp - timeLaunched) / SECONDS_IN_A_DAY;

    if (daySinceLaunched > numdays) {
        d = daySinceLaunched - numdays;
    }
    for (; d <= daySinceLaunched; d++) {
        boughtCount += tokensBoughtXDays[d];
    }
    return boughtCount;
}

function getTokensSoldXDays() public view returns (uint256) {
    if (timeLaunched == 0) return tokensSoldXDays[0];

    uint256 soldCount = 0;
    uint256 d;
    uint256 numdays = daysInCalculation / SECONDS_IN_A_DAY;
    uint256 daySinceLaunched = (block.timestamp - timeLaunched) / SECONDS_IN_A_DAY;

    if (daySinceLaunched > numdays) {
        d = daySinceLaunched - numdays;
    }
    for (; d <= daySinceLaunched; d++) {
        soldCount += tokensSoldXDays[d];
    }
    return soldCount;
}
```

Audit Findings

WandInvestments

WND-11 - Logical Medium Severity

Resolution

The team has resolved this issue.

These functions now correctly account for the trading volume on the same day.

Audit Findings

WandInvestments

WND-12 - Logical Informational Severity

When locking funds using the cashOutScepter function, the sptrTreasuryBal state variable gets updated in that same transaction before the user calls the claimLockedUSD function. This results in the value of sptrTreasuryBal not being in sync with the actual Scepter treasury balance.

This can result in confusion when there are externel revenue streams to the Scepter treasury balance (or losses), which would require calling the updateSPTRTreasuryBal function.

Recommendation

To avoid this confusion, add a new function, addOrSubFromSPTRTreasuryBal, that receives a signed integer that it adds to the sptrTreasuryBal, which can be used to simply add external revenues to the Scepter treasury (or subtract losses) without having to keep track of the locked funds:

```
function addOrSubFromSPTRTreasuryBal(int256 amount) external onlyOwner {
    if (amount < 0) {
        scepterData.sptrTreasuryBal -= uint256(-amount) * DECIMALS;
    } else {
        scepterData.sptrTreasuryBal += uint256(amount) * DECIMALS;
    }
    calcSPTRData();
}
```

Resolution

The team has resolved this issue.

The Scepter treasury balance is only updated when the user receives the funds in the ClaimLockedUSD function, completely avoiding this issue.

Audit Findings

WandInvestments

WND-13 - Logical Informational Severity

This finding compiles smaller informational recommendations into a single recomendation for a more concise and digestable report.

Note: the full code with all of the recommendations will be sent separately.

Recommendations

- Move the seed amounts to constant state variables
- Change the SCEPTER_TREASURY_ADDR, BATON_TREASURY_ADDR, and DEV_WALLET_ADDR state variables to constant
- Delete the unused riskTreasuryAddr, maxWithdrawTax, daysTax, tokenStable, airdroppedHistory, btonHolders, and airDropAmtPerBton state variables
- Change the SPTR, WAND, and BTON state variables to constant and move their values from the constructor to where the variables are declared
- Delete the taxForLocks state variable, and replace with simple math where it is being used in the cashOutScepter function
- Remove the payable modifier from the cashOutBaton and transformScepterToBaton functions
- Default sptrGrowthFactor and sptrSellFactor to 3e17 if getCircSupplyXDays() returns 0
- Supply the Scepter and Baton treasury values to the Launch function instead of hardcoding them since their values might not be known at contract deployment time
- Delete the unused calcBatonAirdropRate function

Audit Findings

WandInvestments

WND-13 - Logical Informational Severity

Resolution

The team has resolved this issue.

The recommended code change has been applied.

Overview

WandInvestments

- This contract provides the functionality to allow users to buy and sell Scepter and Baton tokens, setting their price according to the calculations described in the whitepaper. It also allows users to transform Scepter tokens into Baton tokens.

- The adminDelegator publicly viewable variable contains the address that is allowed to call the updateSPTRTreasuryBal, updateBTONTreasuryBal, and addWhitelistee, and Launch functions. The contract owner has admin powers to change this address.

- The cashOutScepter function allows users to sell their Scepter tokens. They can either receive 9.5% of the selling price straight away, or to lock their funds for up to 9 days. Each additional lock day grants them an extra 9.5% of the selling price, up to a maximum of 95% of the selling price at 9 locked days. The locked funds can be claimed once the full lock duration has passed using the ClaimLockedUSD function. If a user locks funds when they already have unclaimed locked funds, the new locked duration and amount are added to the previous lock, and the user would have to wait for the added lock duration before claiming any funds. The user can choose which token they would like to claim from a set of supported tokens that can be found in the stableERC20Info publicly viewable variable. 0.5% of the selling price is sent to the DEV_WALLET_ADDR if the user decides not to lock their funds.

- The cashOutBaton function allows users to sell their Baton tokens. The selling price is the smaller amount between half of the Scepter backing price and 30% of the Baton backing price, as detailed in the whitepaper. The user can choose which token they would like to claim from a set of supported tokens that can be found in the stableERC20Info publicly viewable variable.

Overview

WandInvestments

- The transformScepterToBaton function allows users to exchange some of their Scepter tokens for the same amount of Baton tokens. The user can choose which token they would like to be transferred from the SCEPTER_TREASURY_ADDR to the BATON_TREASURY_ADDR from a set of supported tokens that can be found in the stableERC20Info publicly viewable variable. 90% of the Scepter backing price is transferred in the chosen token.

- The buyScepter function allows users to buy Scepter tokens. The function can only be used after 48 hours have passed from the Launch function being called. The user can choose which token they would like to purchase with from a set of supported tokens that can be found in the stableERC20Info publicly viewable variable. Users can buy a maximum of 250,000 Scepter tokens in a single buy transaction. 95% of the buy price is sent to the SCEPTER_TREASURY_ADDR and the remaining 5% is sent to the DEV_WALLET_ADDR.

- The wlBuyScepter function allows users to buy Scepter tokens. Only whitelisted users can use this functions, and they can only purchase a maximum of 2000, 1500, or 1000 Scepter tokens depending on ther whitelist tier. This function can only be used in the first 48 hours from the Launch function being called. The price is set at a fixed $1 per token. The user can choose which token they would like to purchase with from a set of supported tokens that can be found in the stableERC20Info publicly viewable variable. 95% of the buy price is sent to the SCEPTER_TREASURY_ADDR and the remaining 5% is sent to the DEV_WALLET_ADDR.

- The claimLockedUSD function allows users to claim any locked funds they have from the cashOutScepter function after the lock duration has passed. The user can choose which token they would like to claim from a set of supported tokens that can be found in the stableERC20Info publicly viewable variable. 95% of the claimed amount is sent to the SCEPTER_TREASURY_ADDR and the remaining 5% is sent to the DEV_WALLET_ADDR.

Overview

WandInvestments

- The turnOnOffTrading function can be used to enable or disable the cashOutScepter, cashOutBaton, transformScepterToBaton, buyScepter, wlBuyScepter, and claimLockedUSD functions. The contract owner has admin powers to call this function.

- The updateSPTRTreasuryBal function can be used to update the value of the Scepter treasury balance in the smart contract, which would affect the buying and selling prices of the Scepter tokens, and could affect the selling price of the Baton tokens. The adminDelegator address has admin powers to call this function.

- The addOrSubFromSPTRTreasuryBal function can be used to update the value of the Scepter treasury balance in the smart contract, which would affect the buying and selling prices of the Scepter tokens, and could affect the selling price of the Baton tokens. The contract owner has admin powers to call this function.

- The updateBTONTreasuryBal function can be used to update the value of the Baton treasury balance in the smart contract, which could affect the selling price of the Baton tokens. The adminDelegator address has admin powers to call this function.

- The Launch function can be used to mint the initial seed amounts of Scepter and send them to distribution contracts, as well as initialize some values in the smart contract, including the buy and sell prices of Scepter. It also enables the cashOutScepter, cashOutBaton, transformScepterToBaton, buyScepter, wlBuyScepter, and claimLockedUSD functions. The adminDelegator address has admin powers to call this function.

- The setDaysUsedInFactors function can be used to change the number of days that are used the calculations that determine the buy and sell prices of Scepter. It could also affect the sell price of Baton. The default number of days used in those calculations is five. The contract owner has admin powers to call this function.

Overview

WandInvestments

- The addWhitelistee function can be used to whitelist an address to be allowed to use the wlBuyScepter function. The adminDelegator address has admin powers to call this function.

- The addStable function can be used to add or remove support for tokens that can be used for buying and selling Scepter and Baton. The contract owner has admin powers to call this function.

Audit Findings

mathFuncs

MATH-01 - Logical Medium Severity

The decMul18 and decDiv18 functions are:

(x * y + DECIMALS / 2) / DECIMALS and (x * DECIMALS + y / 2) / y respectively. The addition of DECIMALS / 2 and y / 2 is incorrect as it would result in an offset if the arguments are small enough.

Recommendation

Change those two functions to x * y / DECIMALS and x * DECIMALS / y respectively:

```
function decMul18(uint x, uint y) internal pure returns (uint decProd) {
    decProd = x * y / DECIMALS;
}

function decDiv18(uint x, uint y) internal pure returns (uint decQuotient) {
    require(y != 0, "Division by zero");
    decQuotient = x * DECIMALS / y;
}
```

Audit Findings

mathFuncs

MATH-02 - Logical Informational Severity

The mul, div, sub, and add functions are not required when the Solidity compilar is 0.8.0 or higher, which introduces automatic revert on underflow and overflow arithmetic operations (https://docs.soliditylang.org/en/v0.8.13/080-breaking-changes.html# silent-changes-of-the-semantics).

Recommendation

Delete these functions and replace their uses with the corresponding arithmetic operations (*, /, -, and + respectively).

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