KINGX Security Review

February 14, 2024

Conducted by:
Blckhv, Independent Security Researcher
Slavcheww, Independent Security Researcher

Contents

1 About SBSecurity

SBSecurity is a company 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.

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

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

4 Executive Summary

The KINGX token is a key element of the free-market economy of the TitanxMarket project, operating within the TitanX ecosystem. One of the significant functions of the KINGX token is to reduce the supply of TitanX tokens, contributing to the increase in the value of the project. By exclusively minting the KINGX token using TITANX tokens, they limit the current and future supply of TITANX.

Overview

Project	KINGX
Repository	https://github.com/daniel-koziara/kingx
Commit Hash	b14b03267a66d9ae518ebd0f5f246f94420b4b01
Date	February 13 – February 14, 2024

Scope

contracts/KingX.sol

Issues Found

Critical Risk	2
High Risk	0
Medium Risk	3
Low/Info Risk	10

5 Findings

5.1 Critical severity

5.1.1 KINGX will not receive fees, when transfer tokens through Uniswap pools

Severity: Critical Risk

Context: KingX.sol#L58, L77

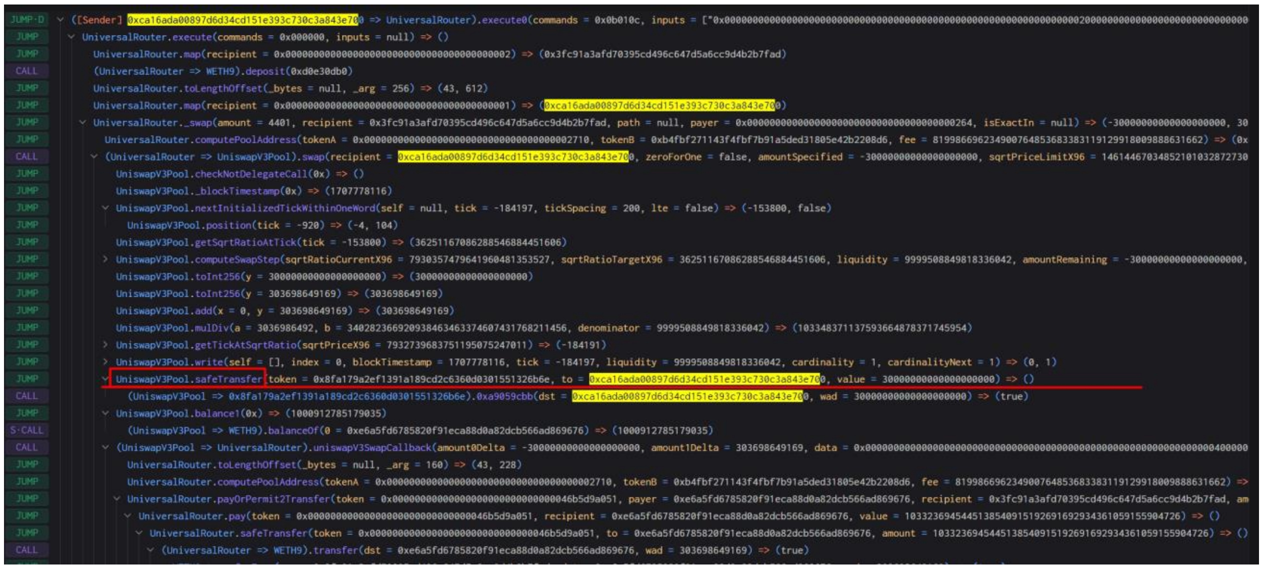
Description: When `KINGX` is transferred through a Uniswap pool, 1% of it needs to go back to the `KINGX` address and then redeemed by the owners. However, there's a mistake in the way `KINGX` checks for the `to` and `from` addresses in `transfer/transferFrom` functions, it incorrectly checks for Uniswap router. Actually, when a token is swapped within Uniswap, it goes from the pool to the user. This means that the concept of fees from Uniswap usage isn't developed as intended.

```
function transfer(
    address to,
    uint256 value
) public override returns (bool) {
    uint256 valueAfterTax = value;

    if (to == routerAddress) { // @audit msg.sender == Uniswap pool
        uint256 taxFee = calculateTaxFee(value);
        valueAfterTax = value - taxFee;
        genesis[GenesisTokens.KINGX] += taxFee;
        super.transfer(taxFeeAddress, taxFee);
    }

    super.transfer(to, valueAfterTax);
    return true;
}
```

Proof of Concept: When a KINGX pool is created, all future swaps will originate from the pool.



```
JUMP-D  ([Sender] 0xca16ada00897d6d34cd151e393c730c3a843e700 => UniversalRouter).execute0(commands = 0x0b010c, inputs = ["0x0000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000
JUMP     UniversalRouter.execute(commands = 0x000000, inputs = null) => ()
JUMP       UniversalRouter.map(recipient = 0x0000000000000000000000000000000000000002) => (0x3fc91a3afd70395cd496c647d5a6cc9d4b2b7fad)
CALL       (UniversalRouter => WETH9).deposit(0xd0e30db0)
JUMP       UniversalRouter.toLengthOffset(_bytes = null, _arg = 256) => (43, 612)
JUMP       UniversalRouter.map(recipient = 0x0000000000000000000000000000000000000001) => (0xca16ada00897d6d34cd151e393c730c3a843e700)
JUMP       UniversalRouter._swap(amount = 4401, recipient = 0x3fc91a3afd70395cd496c647d5a6cc9d4b2b7fad, path = null, payer = 0x0000000000000000000000000000000000000264, isExactIn = null) => (-3000000000000000000, 30
JUMP         UniversalRouter.computePoolAddress(tokenA = 0x0000000000000000000000000000000000002710, tokenB = 0xb4fbf271143f4fbf7b91a5ded31805e42b2208d6, fee = 81998669623490076485368338311912991800988631662) => (0x
CALL         (UniversalRouter => UniswapV3Pool).swap(recipient = 0xca16ada00897d6d34cd151e393c730c3a843e700, zeroForOne = false, amountSpecified = -3000000000000000000, sqrtPriceLimitX96 = 1461446703485210103287273
JUMP           UniswapV3Pool.checkNotDelegateCall(0x) => ()
JUMP           UniswapV3Pool._blockTimestamp(0x) => (1707778116)
JUMP           UniswapV3Pool.nextInitializedTickWithinOneWord(self = null, tick = -184197, tickSpacing = 200, lte = false) => (-153800, false)
JUMP             UniswapV3Pool.position(tick = -920) => (-4, 104)
JUMP           UniswapV3Pool.getSqrtRatioAtTick(tick = -153800) => (36251167086288546884451606)
JUMP           UniswapV3Pool.computeSwapStep(sqrtRatioCurrentX96 = 7930357479641960481353527, sqrtRatioTargetX96 = 36251167086288546884451606, liquidity = 9999508849818336042, amountRemaining = -3000000000000000000,
JUMP           UniswapV3Pool.toInt256(y = 3000000000000000000) => (3000000000000000000)
JUMP           UniswapV3Pool.toInt256(y = 303698649169) => (303698649169)
JUMP           UniswapV3Pool.add(x = 0, y = 303698649169) => (303698649169)
JUMP           UniswapV3Pool.mulDiv(a = 3036986492, b = 340282366920938463463374607431768211456, denominator = 9999508849818336042) => (103348371137593664878371745954)
JUMP           UniswapV3Pool.getTickAtSqrtRatio(sqrtPriceX96 = 7932739683751195075247011) => (-184191)
JUMP           UniswapV3Pool.write(self = [], index = 0, blockTimestamp = 1707778116, tick = -184197, liquidity = 9999508849818336042, cardinality = 1, cardinalityNext = 1) => (0, 1)
JUMP           UniswapV3Pool.safeTransfer(token = 0x8fa179a2ef1391a189cd2c6360d0301551326b6e, to = 0xca16ada00897d6d34cd151e393c730c3a843e700, value = 3000000000000000000) => ()
CALL             (UniswapV3Pool => 0x8fa179a2ef1391a189cd2c6360d0301551326b6e).0xa9059cbb(dst = 0xca16ada00897d6d34cd151e393c730c3a843e700, wad = 3000000000000000000) => (true)
JUMP           UniswapV3Pool.balance1(0x) => (1000912785179035)
S-CALL           (UniswapV3Pool => WETH9).balanceOf(0 = 0xe6a5fd6785820f91eca88d0a82dcb566ad869676) => (1000912785179035)
CALL           (UniswapV3Pool => UniversalRouter).uniswapV3SwapCallback(amount0Delta = -3000000000000000000, amount1Delta = 303698649169, data = 0x000000000000000000000000000000000000000000000000000000000000400000
JUMP             UniversalRouter.toLengthOffset(_bytes = null, _arg = 160) => (43, 228)
JUMP             UniversalRouter.computePoolAddress(tokenA = 0x0000000000000000000000000000000000002710, tokenB = 0xb4fbf271143f4fbf7b91a5ded31805e42b2208d6, fee = 81998669623490076485368338311912991800988631662) =>
JUMP             UniversalRouter.payOrPermit2Transfer(token = 0x00000000000000000000000000000046b5d9a051, payer = 0xe6a5fd6785820f91eca88d0a82dcb566ad869676, recipient = 0x3fc91a3afd70395cd496c647d5a6cc9d4b2b7fad, am
JUMP               UniversalRouter.pay(token = 0x00000000000000000000000000000046b5d9a051, recipient = 0xe6a5fd6785820f91eca88d0a82dcb566ad869676, value = 1033236945445138540915192691692934361059155904726) => ()
JUMP                 UniversalRouter.safeTransfer(token = 0x00000000000000000000000000000046b5d9a051, to = 0xe6a5fd6785820f91eca88d0a82dcb566ad869676, amount = 1033236945445138540915192691692934361059155904726) => ()
CALL                   (UniversalRouter => WETH9).transfer(dst = 0xe6a5fd6785820f91eca88d0a82dcb566ad869676, wad = 303698649169) => (true)
```

Swap transaction trace:
https://dashboard.tenderly.co/tx/goerli/0x6954d6f4dfc8c2220c9b18ac11f75d7b632af37c8158da0b644170fd1ef3dc89?trace=0.2.0

Recommendation: To address this issue, the code should be modified to verify whether the transfer/transferFrom, msg.sender, to and from addresses represents a valid pool with KINGX as either token0 or token1 during each transfer.

Resolution: Fixed

5.1.2 Wrong percentage fee, when transfer tokens through Uniswap

Severity: Critical Risk

Context: KingX.sol#L15

Description: The fee for Uniswap transfers/swaps should be 1%, but right now it's hardcoded at 10%.

```
uint256 public constant taxFeePercent = 10;
```

In the `calculateTaxFee()` function, it will deduct **10%** from the transferred amount as tax.

```
function calculateTaxFee(uint256 amount) private pure returns (uint256) {
    return (amount * taxFeePercent) / 100;
}
```

If you transfer **10,000e18 KINGX** tokens, the protocol should take a fee of **1%**, which is **100e18KINGX**. However, it will take **1,000e18 KINGX** instead.

Another problem is that the fee is only applied if the transferred amount is greater than 100, but due to the way transactions are priced, the loss of fees will never happen because the transaction will always cost more than the transferred amount in these small cases.

Recommendation: Update the percentage calculation to use basis points instead, and correct the `taxFeePercent` value.

```
uint256 public constant taxFeePercentBps = 100; // 1%
```

```
function calculateTaxFee(uint256 amount) private pure returns (uint256) {
    return (amount * taxFeePercentBps) / 10000;
}
```

Resolution: Fixed

5.2 Medium severity

5.2.1 Network congestion can cause users to receive less KINGX tokens

Severity: Medium Risk

Context: KingX.sol#L94

Description: When users mint `KINGX` tokens, the amount they receive depends on the elapsed hours. Because the `mint()` function doesn't require users to specify a minimum output amount of `KINGX` tokens, the actual amount they receive might be quite different from what they expected.

If a user's transaction is stuck in the Ethereum mempool because they provided a low gas price, and the current hour changes before the transaction is executed when it finally does execute, the user will receive significantly fewer tokens than they initially anticipated.

Recommendation: Consider adding `deadline` and `minOutput` parameters to `mint` function.

Resolution: Acknowledged

5.2.2 Do not expose owners' wallets

Severity: Medium Risk

Context: KingX.sol#L94

Description: The owners' addresses are hardcoded, which isn't the best approach. This makes it easier for hackers to target these addresses and try to hack them to get the rewards.

```
address constant HELLWHALE_OWNER = 0x8add03eafe6E89Cc28726f8Bb91096C2dE139fFb;
address constant DANIEL_KOZIARA_OWNER = 0x7e603e457d8C0D61351111614ad977315Dfc77aa;
address constant KRONOS_OWNER = 0x9FEAcbaf3C4277bC9438759058E9E334f866992a;
```

Another common problem is when one of the owners forgets their account and private key. This means that all their rewards will be locked.

Recommendation: Consider passing their address on deployment and allowing them to redeem their rewards in other wallets, so that the fee distribution is compatible with a pull over push pattern, since now it transfer all fees at once.

Resolution: Acknowledged. Client’s response: *We want to be transparent to community as much as possible.*

5.2.3 Multihop swaps will pay double fees to the owners

Severity: Medium Risk

Context: KingX.sol#L62, L70

Description: `KINGX` gets a 1% fee for all swaps on `Uniswap`. If a swap is routed from more than 2 pools, for example, KINGX → USDC → WETH, the fee will be charged twice.

When a multipool swap is initiated with `KINGX`, you send tokens to `UniswapRouter`. This is where the first fee will be charged. Then, the router sends tokens to the first pool (`KINGX` → token2), where the second fee will be charged.

Recommendation: This issue is difficult to mitigate as this is the way `Uniswap` multipool swaps work, the team can refund the user the second fee if they feel it is necessary.

Resolution: Acknowledged

5.3 Low/Info severity

5.3.1 Use safeTransfer instead of tranfer

Severity: Low Risk

Context: KingX.sol#L112

Description: In the mint(), the TitanX is transferred from the user. The first transfer is done using safeTransfer(), while the other transfer is done using transfer(). buyAndBurn address is passed as a constructor argument and can eventually be address(0) - safeTransfer, unlike transfer has a check that will prevent sending the tokens to the zero address.

```
function mint(uint256 titanXAmount) external {
    require(
        block.timestamp > contractStartTime,
        "KingX_Minting: Minting not allowed yet"
    );

    require(
        block.timestamp <= contractStartTime + MINTING_PERIOD,
        "KingX_Minting: Minting period has ended"
    );

    uint256 genesisAmount = (titanXAmount * 3) / 100;

    titanX.safeTransferFrom(msg.sender, address(this), genesisAmount);
    genesis[GenesisTokens.TITANX] += genesisAmount;

    uint256 transferAmount = titanXAmount - genesisAmount;

    titanX.transferFrom(msg.sender, buyAndBurnAddress, transferAmount); // @audit safeTransferFrom
```

Recommendation: Change to safeTransfer.

Resolution: Fixed

5.3.2 Minting will start later than expected

Severity: Low Risk

Context: KingX.sol#L96

Description: KingX minting is supposed to start 1 hour after it has been deployed, but due to the wrong equality check it will start 1 sec later.

```
function mint(uint256 titanXAmount) external {
    require(
        block.timestamp > contractStartTime,
        "KingX_Minting: Minting not allowed yet"
    );
```

Recommendation: Add equal sign to the require in order code to behave as expected.

Resolution: Fixed

5.3.3 Zero address checks are missing

Severity: Low Risk

Context: KingX.sol#L56-57

Description: `buyAndBurnAddress` lacks a zero address check. Additionally, normal transfer is used which will not revert if the recipient is `address(0)`. This will have a direct economic impact on the protocol because the TitanX ecosystem relies on the `buyAndBurn` functionality.

```
constructor(address _buyAndBurnAddress, address _initialLpAddress) ERC20("KINGX", "KINGX") {
    titanX = IERC20(TITANX_ADDRESS);
    buyAndBurnAddress = _buyAndBurnAddress; //@audit lacks 0 address check
    initialLpAddress = _initialLpAddress; //mint will revert if it is address 0
    contractStartTime = block.timestamp + 1 hours;
    taxFeeAddress = address(this);
    _mint(initialLpAddress, 20e9 * 1e18);
}
```

Recommendation: Add zero address check to verify that there are no important contract addresses left uninitialized.

Resolution: Fixed

5.3.4 Do not use hardcoded addresses for external contracts

Severity: Low Risk

Context: KingX.sol#L30, 32

Description: `KINGX` uses `Uniswap Router` and `TITANX` token addresses. But it's better not to hardcode them because these addresses could change in the future due to reasons like hacks or upgrades.

```
address public constant routerAddress =
        0x3fC91A3afd70395Cd496C647d5a6CC9D4B2b7FAD; // universal router uniswap

address constant TITANX_ADDRESS =
        0xF19308F923582A6f7c465e5CE7a9Dc1BEC6665B1;
```

Recommendation: Include these addresses in the constructor and add owner functions to change them if needed.

Resolution: Fixed

5.3.5 Accidentally sent tokens cannot be refunded

Severity: Low Risk

Context: KingX.sol#

Description: Accidentally sent tokens to `KingX` cannot be withdrawn and will remain locked, there is a function to withdraw the rewards allocated from the owners, but it relies on internal accounting. If someone sends tokens directly to the `KingX` address they will remain locked.

Recommendation: Add a skim function to retrieve any mistakenly sent tokens.

Resolution: Fixed

5.3.6 Exchange rate equation can be simplified

Severity: Informational Risk

Context: KingX.sol#115-116

Description: When calculating the mint rate of the `KingX` token `fullHours` variable is unnecessary and only adds confusion to the reader.

Recommendation: Modify the equation to make it clear why there is division and then multiplication to 1 hour.

Resolution: Fixed

5.3.7 `Context` is not used anywhere

Severity: Informational Risk

Context: KingX.sol#6, 8

Description: `KingX` inherits `Context` from OZ but doesn't use any of its functions. Also, there is no need to use its functionality, using the `msg.sender` in a normal way is better in this context.

Recommendation: Remove `Context` from the `KingX's` inherited contracts.

Resolution: Fixed

5.3.8 `KINGX` table shows wrong information

Severity: Informational Risk

Context: Documentation

Description: `KINGX` documentation contains a table explaining the correlation between hours since minting has started and the mint ratio, but it contains wrong information and can lead to confusion for the reader. https://docs.titanxmarket.win/kingx-token

Recommendation: Modify the table to correspond to the actual hours, also the right mint ratio should be added.

Resolution: Fixed

5.3.9 Events not emitted in important functions

Severity: Informational Risk

Context: KingX.sol#133

Description: `distributeGenesisRewards` is an important function that should emit events in order for the code to behave consistently. Proper event emission is important for the off-chain monitoring tools. Also, this function can be called by anyone and it is a good approach to show information, for example, the function name and how many tokens each participant has received, that will give context to the `transfer` event emitted.

Recommendation: Create and use a new event to indicate when `distributeGenesisRewards` is called.

Resolution: Fixed

5.3.10 Inconsistent code formatting

Severity: Informational Risk

Context: KingX.sol

Description: All the contracts need to follow the formatting rules defined by the Solidity Style Guide.

Resolution: Fixed