

MixBytes()

OKX xAsset Security Audit Report

FEBRUARY 02, 2026

Table of Contents

1. Introduction	2
1.1 Disclaimer	2
1.2 Executive Summary	2
1.3 Project Overview	3
1.4 Security Assessment Methodology	5
1.5 Risk Classification	7
1.6 Summary of Findings	8
2. Findings Report	9
2.1 Critical	9
2.2 High	9
2.3 Medium	9
2.4 Low	9
L-1 Allowance Reset During Configuration Update	9
L-2 No Two-Step Role Transfer Pattern	10
L-3 Unrestricted Role and Ownership Renunciation	11
L-4 Exchange Rate Freshness Not Stored	12
L-5 Non-Existent Caller Handling Issues	13
L-6 Rate Limits Use Absolute Deltas	14
L-7 Initial Exchange Rate is Zero	15
L-8 Code Quality Improvements	16
3. About MixBytes	17

1. Introduction

1.1 Disclaimer

The audit makes no statements or warranties regarding the utility, safety, or security of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other claims about the fitness of the contracts for a particular purpose or their bug-free status.

1.2 Executive Summary

OKX xAssets is a suite of OKX-backed ERC-20 tokens on EVM chains, including wrapped tokens (`xBTC`, `xETH`, `xsOL`) with 1:1 backing and liquid staking tokens (`xBETH`, `xOKSOL`) that accrue rewards through exchange rate appreciation. The protocol uses role-based access control with restricted minting (only to `authorizedReceiver`), deny list functionality, and pausability. For staking tokens, the `ExchangeRateUpdater` contract enforces rate-limited exchange rate updates to prevent abnormal changes, while `StakedTokenV1` stores the exchange rate in dedicated storage slot. The system bridges OKX's centralized exchange with EVM chains.

The audit was carried out over a period of 2 days by a team of 3 auditors, combining manual review with automated tooling.

During the audit, the following areas were checked:

Proxy and Upgradeability: `Proxy.sol` and `xbtcProxy.sol` are minimal wrappers over OpenZeppelin `TransparentUpgradeableProxy` without additional logic or state. Admin/user separation is enforced by the `ProxyAdmin` pattern – regular users cannot call proxy admin functions. Implementation contracts (`xbtc`, `xToken`) correctly call `_disableInitializers()` in constructor, preventing direct initialization of the implementation.

DenyList Bypass Prevention: The `_update()` function in `xbtc` and `xToken` contracts checks three addresses: `from`, `to`, and `msg.sender`. This prevents denylist bypass via `transferFrom` – even if `sender/receiver` are clean, a blocked spender cannot execute the transaction.

Dual Role Assignment to denyLister: In `xbtc.sol` at lines 92–93, the `denyLister` address receives both `DEFAULT_ADMIN_ROLE` and `DENY_LISTER_ROLE` during initialization, whereas in the `xToken.sol` these roles are separated between `admin` and `denyLister` parameters. The `DEFAULT_ADMIN_ROLE` grants the ability to manage all roles including granting/revoking `MINTER_ROLE` and `DENY_LISTER_ROLE` itself. Assigning both roles to a single address may not align with the principle of separation of duties.

Staking logic behind the underlying tokens, fund rescue mechanisms, off-chain backend systems, and the actual reserve backing verification were not part of this audit scope.

The codebase demonstrates a solid foundation with well-structured contracts and appropriate use of OpenZeppelin libraries. We recommend paying additional attention to exchange rate

limiting, role management safety, and proper implementation of checks for zero addresses and non-existent entities.

1.3 Project Overview

Summary

Title	Description
Client	OKX
Category	CEX
Project	xAsset
Type	Solidity
Platform	EVM
Timeline	16.01.2026 – 22.01.2026

Scope of Audit

File	Link
<code>contracts/ExchangeRateUtil.sol</code>	ExchangeRateUtil.sol
<code>contracts/ExchangeRateUpdater.sol</code>	ExchangeRateUpdater.sol
<code>contracts/Proxy.sol</code>	Proxy.sol
<code>contracts/xbtc.sol</code>	xbtc.sol
<code>contracts/StakedTokenV1.sol</code>	StakedTokenV1.sol
<code>contracts/xbtcProxy.sol</code>	xbtcProxy.sol
<code>contracts/xToken.sol</code>	xToken.sol
<code>contracts/RateLimit.sol</code>	RateLimit.sol

Versions Log

Date	Commit Hash	Note
16.01.2026	5867d201df09784b2935ae6095a50ac410619969	Initial Commit
22.01.2026	0a89809bc2ec49a0cc8054550ef22769a89dd989	Commit for Re-audit

Mainnet Deployments

`xbtcProxy.sol` (`0xb7C00000bcDEeF966b20B3D884B98E64d2b06b4f`) was deployed for `xbtc.sol`
`Proxy.sol` (`0xE7B000003A45145decf8a28FC755aD5eC5EA025A`) was deployed for `xETH.sol`
`Proxy.sol` (`0x505000008DE8748DBd4422ff4687a4FC9bEba15b`) was deployed for `xSOL.sol`
`Proxy.sol` (`0xAFeab3B85B6A56cF5F02317F0f7A23340eb983D7`) was deployed for `xBETH.sol`
`Proxy.sol` (`0x14a686103854DAB7b8801E31979CAA595835B25d`) was deployed for `xOKSOL.sol`
`ExchangeRateUpdater.sol` (`0xFC85bd0C3d470B3a9d8CDc60b26A238D5bA2D666`) was deployed for `xBETH.sol`
`ExchangeRateUpdater.sol` (`0x6a5A44EB0519C0664E65026c80eB7b4bC3778874`) was deployed for `xOKSOL.sol`

File	Address	Blockchain
<code>xbtcProxy.sol</code>	<code>0xb7C00000bcDEeF966b20B3D884B98E64d2b06b4f</code>	X Layer
<code>xbtc.sol</code>	<code>0xb8AefaceA8B9AFD0F13CbCE33AD35075714C62ef</code>	X Layer
<code>Proxy.sol</code>	<code>0xE7B000003A45145decf8a28FC755aD5eC5EA025A</code>	X Layer
<code>xETH.sol</code>	<code>0x753BA5BAdA3f9C7fF1119C6B93D31747d58d0303</code>	X Layer
<code>Proxy.sol</code>	<code>0x505000008DE8748DBd4422ff4687a4FC9bEba15b</code>	X Layer
<code>xSOL.sol</code>	<code>0xB7B0673E87BB18877E75Dcf84c3cA4f656782d0</code>	X Layer
<code>Proxy.sol</code>	<code>0xAFeab3B85B6A56cF5F02317F0f7A23340eb983D7</code>	X Layer
<code>xBETH.sol</code>	<code>0x3CCdAf695600474A4c4303E05B5eA4922e1117aF</code>	X Layer
<code>ExchangeRateUpdater.sol</code>	<code>0xFC85bd0C3d470B3a9d8CDc60b26A238D5bA2D666</code>	X Layer
<code>Proxy.sol</code>	<code>0x14a686103854DAB7b8801E31979CAA595835B25d</code>	X Layer
<code>xOKSOL.sol</code>	<code>0x26152C9203B214E536a562A5c96988B6C78bf8d5</code>	X Layer
<code>ExchangeRateUpdater.sol</code>	<code>0x6a5A44EB0519C0664E65026c80eB7b4bC3778874</code>	X Layer

1.4 Security Assessment Methodology

Project Flow

Stage	Scope of Work
Interim audit	<p>Project Architecture Review:</p> <ul style="list-style-type: none">• Review project documentation• Conduct a general code review• Perform reverse engineering to analyze the project's architecture based solely on the source code• Develop an independent perspective on the project's architecture• Identify any logical flaws in the design <p>OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS.</p>
	<p>Code Review with a Hacker Mindset:</p> <ul style="list-style-type: none">• Each team member independently conducts a manual code review, focusing on identifying unique vulnerabilities.• Perform collaborative audits (pair auditing) of the most complex code sections, supervised by the Team Lead.• Develop Proof-of-Concepts (PoCs) and conduct fuzzing tests using tools like Foundry, Hardhat, and BOA to uncover intricate logical flaws.• Review test cases and in-code comments to identify potential weaknesses. <p>OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY.</p>
	<p>Code Review with a Nerd Mindset:</p> <ul style="list-style-type: none">• Conduct a manual code review using an internally maintained checklist, regularly updated with insights from past hacks, research, and client audits.• Utilize static analysis tools (e.g., Slither, Mytril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors. <p>OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS.</p>

Stage	Scope of Work
	<p>Consolidation of Auditors' Reports:</p> <ul style="list-style-type: none"> • Cross-check findings among auditors • Discuss identified issues • Issue an interim audit report for client review <p>OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.</p>
Re-audit	<p>Bug Fixing & Re-Audit:</p> <ul style="list-style-type: none"> • The client addresses the identified issues and provides feedback • Auditors verify the fixes and update their statuses with supporting evidence • A re-audit report is generated and shared with the client <p>OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED.</p>
Final audit	<p>Final Code Verification & Public Audit Report:</p> <ul style="list-style-type: none"> • Verify the final code version against recommendations and their statuses • Check deployed contracts for correct initialization parameters • Confirm that the deployed code matches the audited version • Issue a public audit report, published on our official GitHub repository • Announce the successful audit on our official X account <p>OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT.</p>

1.5 Risk Classification

Severity Level Matrix

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- **High** – Theft from 0.5% OR partial/full blocking of funds (>0.5%) on the contract without the possibility of withdrawal OR loss of user funds (>1%) who interacted with the protocol.
- **Medium** – Contract lock that can only be fixed through a contract upgrade OR one-time theft of rewards or an amount up to 0.5% of the protocol's TVL OR funds lock with the possibility of withdrawal by an admin.
- **Low** – One-time contract lock that can be fixed by the administrator without a contract upgrade.

Likelihood

- **High** – The event has a 50–60% probability of occurring within a year and can be triggered by any actor (e.g., due to a likely market condition that the actor cannot influence).
- **Medium** – An unlikely event (10–20% probability of occurring) that can be triggered by a trusted actor.
- **Low** – A highly unlikely event that can only be triggered by the owner.

Action Required

- **Critical** – Must be fixed as soon as possible.
- **High** – Strongly advised to be fixed to minimize potential risks.
- **Medium** – Recommended to be fixed to enhance security and stability.
- **Low** – Recommended to be fixed to improve overall robustness and effectiveness.

Finding Status

- **Fixed** – The recommended fixes have been implemented in the project code and no longer impact its security.
- **Partially Fixed** – The recommended fixes have been partially implemented, reducing the impact of the finding, but it has not been fully resolved.
- **Acknowledged** – The recommended fixes have not yet been implemented, and the finding remains unresolved or does not require code changes.

1.6 Summary of Findings

Findings Count

Severity	Count
Critical	0
High	0
Medium	0
Low	8

Findings Statuses

ID	Finding	Severity	Status
L-1	Allowance Reset During Configuration Update	Low	Acknowledged
L-2	No Two-Step Role Transfer Pattern	Low	Acknowledged
L-3	Unrestricted Role and Ownership Renunciation	Low	Acknowledged
L-4	Exchange Rate Freshness Not Stored	Low	Acknowledged
L-5	Non-Existent Caller Handling Issues	Low	Acknowledged
L-6	Rate Limits Use Absolute Deltas	Low	Acknowledged
L-7	Initial Exchange Rate is Zero	Low	Acknowledged
L-8	Code Quality Improvements	Low	Fixed

2. Findings Report

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

L-1	Allowance Reset During Configuration Update		
Severity	Low	Status	Acknowledged

Description

In the `RateLimit.configureCaller()` function at line 123, the `allowances[caller]` is directly set to the new `amount` without first calling `_replenishAllowance()`. This means if a caller has already spent part of their allowance and the owner reconfigures them, the spent amount is effectively refunded. A malicious caller could exploit this by monitoring the mempool for `configureCaller()` transactions and frontrunning them to spend their old allowance, then benefit from the reset allowance, effectively exceeding both the old and new `maxAllowance` values.

Recommendation

We recommend calling `_replenishAllowance()` first in the `configureCaller()` function before updating `allowances[caller]`, then capping it at the new `amount` if it exceeds the new limit. Additionally, we recommend implementing separate `increaseAllowance()` and `decreaseAllowance()` functions instead of directly resetting the allowance to prevent frontrunning attacks.

Client's Commentary:

Acknowledged. This is the expected behaviour. Both owner and caller are controlled by OKX. When allowance is too small, OKX call `configureCaller()` to immediately increase allowance.

L-2	No Two-Step Role Transfer Pattern		
Severity	Low	Status	Acknowledged

Description

The `xToken.transferMinter()` function at line 260 and `xToken.transferDenyLister()` function at line 283 immediately revoke the role from the current holder and grant it to the new address in a single transaction. If the `newMinter` or `newDenyLister` address is incorrectly specified (e.g., a typo, wrong address, or contract that cannot handle the role), the role will be permanently lost, potentially bricking critical contract functionality like minting and burning operations.

Recommendation

We recommend implementing a two-step transfer pattern where the new role holder must explicitly call an acceptance function to claim the role before the previous holder loses it.

Client's Commentary:

Acknowledged. We use an internal approval process to ensure the correct address is used. Also, `DEFAULT_ADMIN_ROLE` can grant a new role holder.

L-3	Unrestricted Role and Ownership Renunciation		
Severity	Low	Status	Acknowledged

Description

The `xToken` contract inherits from `AccessControlUpgradeable`, which allows any role holder to call `renounceRole()` to renounce their own role, including `DEFAULT_ADMIN_ROLE`. If the admin accidentally or maliciously renounces this role, critical functions like `updateOracle()` in `StakedTokenV1` (line 76) become permanently inaccessible. Additionally, `RateLimit` and `ExchangeRateUpdater` contracts inherit from `Ownable`, exposing `renounceOwnership()`. If the owner renounces ownership, functions like `configureCaller()` and `removeCaller()` become permanently unavailable, freezing the system's ability to manage callers while existing callers retain their allowances indefinitely.

Recommendation

We recommend overriding the `renounceRole()` function in `xToken` to prevent renunciation of `DEFAULT_ADMIN_ROLE`, and overriding or removing `renounceOwnership()` in `RateLimit` and `ExchangeRateUpdater` contracts to prevent accidental ownership loss.

Client's Commentary:

Acknowledged. We use an internal approval process to prevent such accidents.

L-4	Exchange Rate Freshness Not Stored		
Severity	Low	Status	Acknowledged

Description

In `StakedTokenV1.updateExchangeRate()` at line 96, the new exchange rate is stored directly in the `_EXCHANGE_RATE_POSITION` storage slot without recording any timestamp. The `ExchangeRateUpdated` event at line 105 also only emits the oracle address and rate value. This makes it impossible for external consumers or monitoring systems to determine how stale the exchange rate data is, potentially leading to decisions based on outdated information.

Recommendation

We recommend adding a timestamp storage variable that records `block.timestamp` whenever `updateExchangeRate()` is called, and exposing this value through a view function to enable freshness verification.

Client's Commentary:

Acknowledged. We have a backend system that updates the exchange rate routinely, and external consumers can monitor the `ExchangeRateUpdated` event

L-5	Non-Existent Caller Handling Issues		
Severity	Low	Status	Acknowledged

Description

The `RateLimit.currentAllowance()` function at line 160 can be called for any address, including non-existent or removed callers. It unconditionally calls `_replenishAllowance()` at line 161, which at line 170 checks `allowances[caller] == maxAllowances[caller]`. For removed callers where both values are 0, this condition passes and `allowancesLastSet[caller]` gets updated. Also, `estimatedAllowance()` at line 147 calls `_getReplenishAmount()` at line 152, which performs division by `intervals[caller]` at line 204. For non-existent callers where `intervals[caller]` is 0, this causes a division by zero revert instead of a descriptive revert message.

Recommendation

We recommend adding a `require(callers[caller], "RateLimit: caller not configured")` check at the beginning of `currentAllowance()` and `estimatedAllowance()` functions to prevent operations on non-existent callers with clear error messages.

Client's Commentary:

Acknowledged. These two functions are only used internally, and all inputs are guaranteed to be configured callers.

L-6	Rate Limits Use Absolute Deltas		
Severity	Low	Status	Acknowledged

Description

The `ExchangeRateUpdater.updateExchangeRate()` function uses absolute value changes to enforce rate limits. At line 108-113, the `exchangeRateChange` is calculated as the absolute difference between `_newExchangeRate` and `currentExchangeRate`, then compared against `allowances[msg.sender]` at line 116. This means a change from `1e18` to `1.1e18` (10% increase) consumes the same allowance as a change from `10e18` to `10.1e18` (1% increase), even though the economic impact differs significantly. While administrators can configure different `maxAllowances` for different token contracts, this absolute delta approach is uncommon and may not scale proportionally across tokens with vastly different exchange rate magnitudes.

Recommendation

We recommend considering implementing a percentage-based rate limit mechanism that calculates `exchangeRateChange` as a proportion of the current exchange rate, ensuring consistent relative impact regardless of the exchange rate's absolute value.

Client's Commentary:

Acknowledged. We configure different allowances for different tokens and adjust them if needed.

L-7	Initial Exchange Rate is Zero		
Severity	Low	Status	Acknowledged

Description

The `StakedTokenV1.exchangeRate()` function at line 124 returns the value stored in `_EXCHANGE_RATE_POSITION` storage slot, which is uninitialized and therefore defaults to 0. The comment at line 122 states "Returns 0 before the oracle's first update," but this contradicts the specification where a 1:1 exchange rate is expected initially. Since there is no initialization logic in the contract's `initialize()` function (inherited from `xToken`), the exchange rate remains 0 until the oracle calls `updateExchangeRate()`. This requires manual coordination and an additional transaction to set the rate to 1e18 (representing 1:1 ratio with 18 decimals). Besides that, it requires a specific (potentially large for normal operation) `maxAllowance` value being set initially, to allow such exchange rate change.

Recommendation

We recommend either initializing the exchange rate to 1e18 in the `initialize()` function or clearly documenting that the exchange rate starts at 0 and explaining the intended behavior and initialization process.

Client's Commentary:

Acknowledged. We have an atomic deployment script that initializes the exchange rate to 1e18.

L-8	Code Quality Improvements		
Severity	Low	Status	Fixed in 0a89809b

Description

Several code quality issues were identified:

1. `totalSupply() + amount` is calculated twice in `xToken.mint()` at line 141, once for the require check and again implicitly in the error construction.
2. `xToken.removeFromDenyList()` at line 246 does not verify if `denyList[account]` is true before setting it to false, unlike `batchRemoveFromDenyList()` at line 219.
3. `RateLimit.currentAllowance()` is a misleading name for a state-changing function that modifies `allowancesLastSet`.
4. The comment at `ExchangeRateUpdater.updateExchangeRate()` line 84 incorrectly states "`_newExchangeRate` must be less than or equal to the allowance" when it actually checks `exchangeRateChange`.
5. Comment for `RateLimit.allowancesLastSet` at line 53 says "time in seconds since" when it should be "timestamp when".
6. `xToken.initialize()` does not validate `admin`, `denyLister`, or `minter` parameters for zero address, only checking `receiver`.
7. `ExchangeRateUpdater.initialize()` creates confusion by using initialization pattern while inheriting from regular `Ownable` instead of using `Initializable`. Also, this function doesn't emit any initialization event.

Recommendation

We recommend:

- caching the `totalSupply() + amount` calculation in `xToken.mint()`;
- adding `denyList[account]` check in `removeFromDenyList()`;
- renaming the `currentAllowance()` function;
- correcting all misleading comments;
- adding zero address validation for role addresses in `xToken.initialize()`;
- clarifying the initialization approach in `ExchangeRateUpdater`.

Client's Commentary:

4,5,7: Fixed in commit 0a8980. Updated the comments to clarify.

1,2,3,6: Acknowledged.

3. About MixBytes

MixBytes is a leading provider of smart contract audit and research services, helping blockchain projects enhance security and reliability. Since its inception, MixBytes has been committed to safeguarding the Web3 ecosystem by delivering rigorous security assessments and cutting-edge research tailored to DeFi projects.

Our team comprises highly skilled engineers, security experts, and blockchain researchers with deep expertise in formal verification, smart contract auditing, and protocol research. With proven experience in Web3, MixBytes combines in-depth technical knowledge with a proactive security-first approach.

Why MixBytes

- **Proven Track Record:** Trusted by top-tier blockchain projects like Lido, Aave, Curve, and others, MixBytes has successfully audited and secured billions in digital assets.
- **Technical Expertise:** Our auditors and researchers hold advanced degrees in cryptography, cybersecurity, and distributed systems.
- **Innovative Research:** Our team actively contributes to blockchain security research, sharing knowledge with the community.

Our Services

- **Smart Contract Audits:** A meticulous security assessment of DeFi protocols to prevent vulnerabilities before deployment.
- **Blockchain Research:** In-depth technical research and security modeling for Web3 projects.
- **Custom Security Solutions:** Tailored security frameworks for complex decentralized applications and blockchain ecosystems.

MixBytes is dedicated to securing the future of blockchain technology by delivering unparalleled security expertise and research-driven solutions. Whether you are launching a DeFi protocol or developing an innovative dApp, we are your trusted security partner.

Contact Information

-  <https://mixbytes.io/>
-  https://github.com/mixbytes/audits_public
-  hello@mixbytes.io
-  <https://x.com/mixbytes>