# KiKiFinance - stXSAT

## Security Assessment

CertiK Assessed on Apr 22nd, 2025

CertiK Assessed on Apr 22nd, 2025

# KiKiFinance - stXSAT

The security assessment was prepared by CertiK, the leader in Web3.0 security.

## Executive Summary

| TYPES | ECOSYSTEM | METHODS |
|---|---|---|
| DeFi | exSat Network | Formal Verification, Manual Review, Static Analysis |
| **LANGUAGE** | **TIMELINE** | **KEY COMPONENTS** |
| Solidity | Delivered on 04/22/2025 | N/A |

CODEBASE

stxsat

View All in Codebase Page

COMMITS

- 34f529e9e4513bbb0f40e2348bfb60ba88c6cc2e
- 1beb2441b01cc8897ca72d6f985c563e12423401
- d0569f30d2311c94bb31fa0ff8da23169e74b112

View All in Codebase Page

## Highlighted Centralization Risks

- Contract upgradeability
- Fees are unbounded
- Withdraws can be disabled

## Vulnerability Summary

| 19 | 13 | 0 | 6 | 0 |
|---|---|---|---|---|
| Total Findings | Resolved | Partially Resolved | Acknowledged | Declined |

| | | | |
|---|---|---|---|
| 2 | Centralization | 2 Acknowledged | Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets. |
| 0 | Critical | | Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks. |
| 0 | Major | | Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control. |
| 4 | Medium | 3 Resolved, 1 Acknowledged | Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform. |
| 10 | Minor | 7 Resolved, 3 Acknowledged | Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions. |

| | | | |
|---|---|---|---|
| 3 | Informational | 3 Resolved | Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code. |

# TABLE OF CONTENTS | KIKIFINANCE - STXSAT


## Summary

Executive Summary

Vulnerability Summary

Codebase

Audit Scope

Approach & Methods

## Review Notes

Overview

External Dependencies

Addresses

Privileged Functions

## Findings

KIS-05 : Centralized Control of Contract Upgrade

KIS-06 : Centralized Related Risk

KIS-07 : `initialize()` Is Unprotected

KIS-09 : Functions Using `_stakeHelper.deposit()` Are Non-Payable

KIS-23 : Potential XSAT Token Flow Disruption Due to Delayed `syncBuffers` Execution and Validator Capacity Mismatch

KIS-24 : Using `msg.value` Inside a Loop

KIS-08 : Lack of Storage Gap in Upgradeable Contract

KIS-10 : No Upper Limit in `setFee()` function

KIS-11 : `safeApprove()` is deprecated

KIS-12 : Missing Zero Address Validation

KIS-13 : Missing Emit Events

KIS-14 : External Call Inside Loop

KIS-15 : Potential Reentrancy Attack

KIS-16 : Missing Input Validation

KIS-20 : Incompatibility With Deflationary Tokens

KIS-22 : `removeValidators()` Should Check for Duplicate Adjacent Indexes

KIS-17 : Uninitialized Local Variable

KIS-19 : Inconsistency in Reward and Fee Logic in Solidity Implementation

KIS-21 : Discussion on `stXSAT` Setting

## Optimizations

KIS-01 : `For` or `While` Loops May Consume Excessive Gas and Even Lead to DoS

KIS-02 : Use named return values

KIS-03 : State variables that could be declared immutable

## Formal Verification

Considered Functions And Scope

Verification Results

## Appendix

## Disclaimer

# CODEBASE | KIKIFINANCE - STXSAT

## Repository

stxsat

## Commit

- 34f529e9e4513bbb0f40e2348bfb60ba88c6cc2e
- 1beb2441b01cc8897ca72d6f985c563e12423401
- d0569f30d2311c94bb31fa0ff8da23169e74b112

# AUDIT SCOPE | KIKIFINANCE - STXSAT

4 files audited ● 4 files with Acknowledged findings

| ID | Repo | File | SHA256 Checksum |
|---|---|---|---|
| ● BSX | KikiFinance/stxsat-contract | contracts/BaseStXSAT.sol | de9ee33ae3c816874a8f1a32fcd3a7892eab1507a1f01308f803c44a37129bcf |
| ● SXS | KikiFinance/stxsat-contract | contracts/StXSAT.sol | da04c6507cc3c27057adec3df9eff6ad50278eddaaa6f1c7adc7e739406eb456 |
| ● XSA | KikiFinance/stxsat-contract | contracts/XSATStakingRouter.sol | 14a65726a2d612a38bc0e6cd731b2fe4a500c39f133a70b5e6ba5d026cb3cdf1 |
| ● XST | KikiFinance/stxsat-contract | contracts/wstXSAT.sol | 3f8140535edfc5fcc72be45a438cd06321c3c0556a415980d46064691e69b786 |

# APPROACH & METHODS | KIKIFINANCE - STXSAT

This report has been prepared for KiKiFinance to discover issues and vulnerabilities in the source code of the KiKiFinance - stXSAT project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis, Formal Verification, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

# REVIEW NOTES | KIKIFINANCE - STXSAT

## Overview

**KiKiFinance - stXSAT** is a staking and reward distribution system designed for the XSAT token. It includes the following contracts under audit:

- **stXSAT**: A dynamic staking token using a share-based mechanism with rebase support to auto-adjust user balances as rewards accumulate.
- **wstXSAT**: A static wrapper of stXSAT for compatibility with DeFi protocols that do not support rebasing tokens.
- **Staking Router**: Manages multiple validators and handles deposits, withdrawals, and reward claims by interacting with third party contracts (StakeHelper and RewardHelper).

The files currently under review include：

- BaseStXSAT.sol
- StXSAT.sol
- wstXSAT.sol
- XSATStakingRouter.sol

## External Dependencies

In **KiKiFinance - stXSAT**, the module inherits or uses a few of the depending injection contracts or addresses to fulfill the need of its business logic. The scope of the audit treats third party entities as black boxes and assume their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

### Addresses

The following addresses interact at some point with specified contracts, making them an external dependency. All of the following values are initialized either at deployment time or by specific functions in smart contracts.

#### StXSAT

- `xsatToken`
- `stakingRouter`

#### WstXSAT

- `stXSAT`

#### XSATStakingRouter

- `xsat`
- `stXSAT`
- `stakeHelper`
- `rewardHelper`

We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project. **If these contracts or their associated addresses are compromised, or if the system interacts with them incorrectly, it could result in severe consequences for the overall protocol. In particular, if the `stakeHelper` or `rewardHelper` components fail to operate as intended, users who have staked assets may be unable to withdraw their principal or claim their rewards, leading to a loss of user funds and trust in the system.**

Also, the following library/contract are considered as the third-party dependencies:

- @openzeppelin/contracts-upgradeable/
- @openzeppelin/contracts/

## Privileged Functions

In the **KiKiFinance - stXSAT** project, the privileged roles are adopted to ensure the dynamic runtime updates of the project, which are specified in the `Centralization` finding.

The advantage of those privileged roles in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worth noting the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private keys of the privileged accounts are compromised, it could lead to devastating consequences for the project.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

# FINDINGS | KIKIFINANCE - STXSAT

| 19 | 0 | 2 | 0 | 4 | 10 | 3 |
|---|---|---|---|---|---|---|
| Total Findings | Critical | Centralization | Major | Medium | Minor | Informational |

This report has been prepared to discover issues and vulnerabilities for KiKiFinance - stXSAT. Through this audit, we have uncovered 19 issues ranging from different severity levels. Utilizing the techniques of Static Analysis, Formal Verification & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| **KIS-05** | **Centralized Control Of Contract Upgrade** | **Centralization** | **Centralization** | **Acknowledged** |
| **KIS-06** | **Centralized Related Risk** | **Centralization** | **Centralization** | **Acknowledged** |
| KIS-07 | `initialize()` Is Unprotected | Logical Issue | Medium | Resolved |
| KIS-09 | Functions Using `_stakeHelper.deposit()` Are Non-Payable | Denial of Service | Medium | Resolved |
| KIS-23 | Potential XSAT Token Flow Disruption Due To Delayed `syncBuffers` Execution And Validator Capacity Mismatch | Logical Issue | Medium | Acknowledged |
| KIS-24 | Using `msg.value` Inside A Loop | Coding Issue | Medium | Resolved |
| KIS-08 | Lack Of Storage Gap In Upgradeable Contract | Logical Issue | Minor | Resolved |
| KIS-10 | No Upper Limit In `setFee()` Function | Logical Issue | Minor | Acknowledged |
| KIS-11 | `safeApprove()` Is Deprecated | Logical Issue | Minor | Acknowledged |
| KIS-12 | Missing Zero Address Validation | Volatile Code | Minor | Resolved |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| KIS-13 | Missing Emit Events | Inconsistency | Minor | Resolved |
| KIS-14 | External Call Inside Loop | Denial of Service | Minor | Acknowledged |
| KIS-15 | Potential Reentrancy Attack | Concurrency | Minor | Resolved |
| KIS-16 | Missing Input Validation | Volatile Code | Minor | Resolved |
| KIS-20 | Incompatibility With Deflationary Tokens | Volatile Code | Minor | Resolved |
| KIS-22 | `removeValidators()` Should Check For Duplicate Adjacent Indexes | Volatile Code | Minor | Resolved |
| KIS-17 | Uninitialized Local Variable | Coding Issue | Informational | Resolved |
| KIS-19 | Inconsistency In Reward And Fee Logic In Solidity Implementation | Inconsistency | Informational | Resolved |
| KIS-21 | Discussion On `stXSAT` Setting | Logical Issue | Informational | Resolved |

# KIS-05 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

| Category | Severity | Location | Status |
|---|---|---|---|
| Centralization | Centralization | contracts/StXSAT.sol (stxsat): 12; contracts/XSATStakingRouter.sol (stxsat): 16 | Acknowledged |

## Description

In the contracts `StXSAT` and `XSATStakingRouter`, the role `admin` has the authority to update the implementation contracts.

Any compromise to the `admin` account may allow a hacker to take advantage of this authority and change the implementation contract which is pointed by proxy and therefore execute potential malicious functionality in the implementation contract.

## Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

### Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.

- Provide a link to the **medium/blog** with all of the above information included.

### Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

### Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

- Renounce the ownership and never claim back the privileged role;
  OR
- Remove the risky functionality.

*Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.*

## Alleviation

**[KiKiFinance, 04/18/2025]**: Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement.

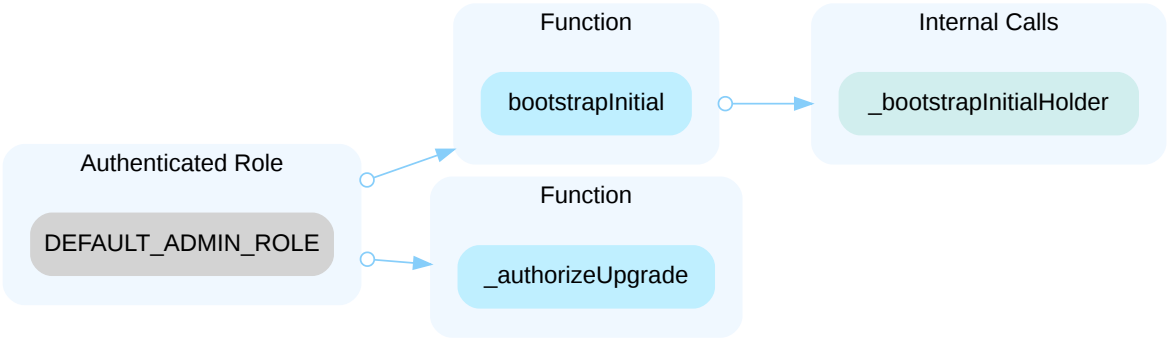
**[CertiK, 04/18/2025]**: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

# KIS-06 | CENTRALIZED RELATED RISK

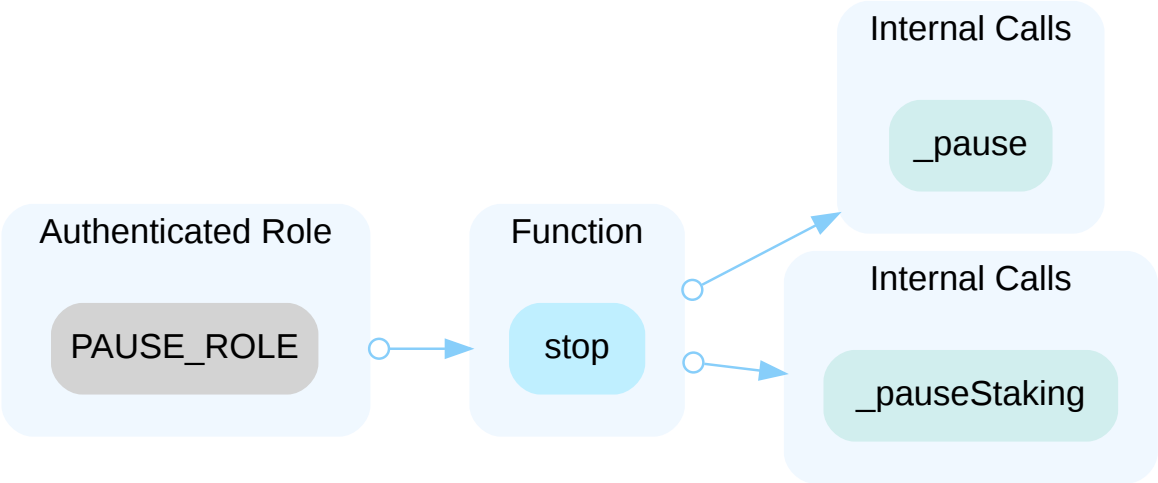
| Category | Severity | Location | Status |
|---|---|---|---|
| Centralization | Centralization | contracts/StXSAT.sol (stxsat): 127, 135, 143, 148, 155, 163, 172; contracts/XSATStakingRouter.sol (stxsat): 101, 104, 114, 122, 166, 175, 347, 352, 363, 400 | Acknowledged |

## Description

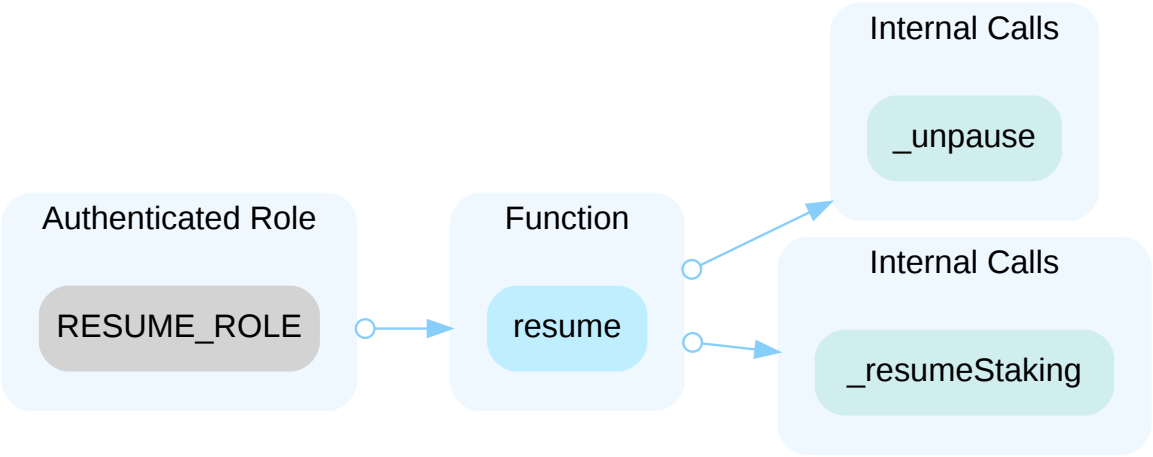
In the contract `StXSAT`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and bootstrap the initial holder, as well as authorize contract upgrades by the admin.



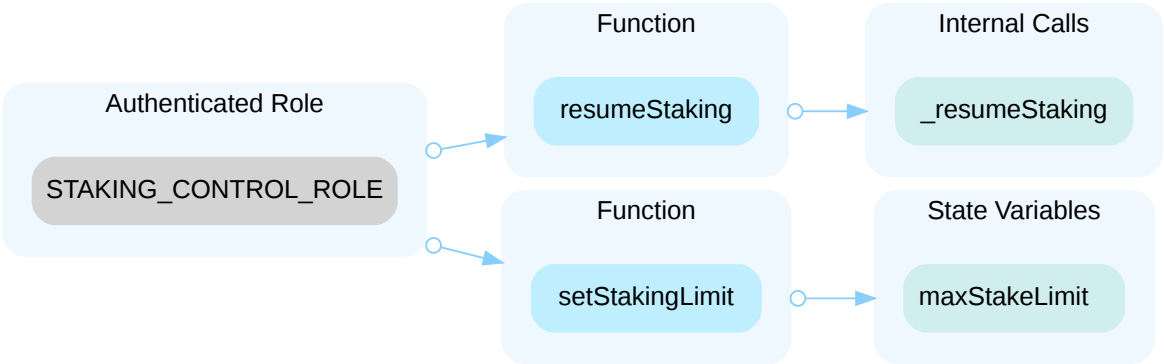

In the contract `StXSAT`, the role `PAUSE_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `PAUSE_ROLE` account may allow the hacker to take advantage of this authority and pause most of the contract functions except for setting contract parameters.



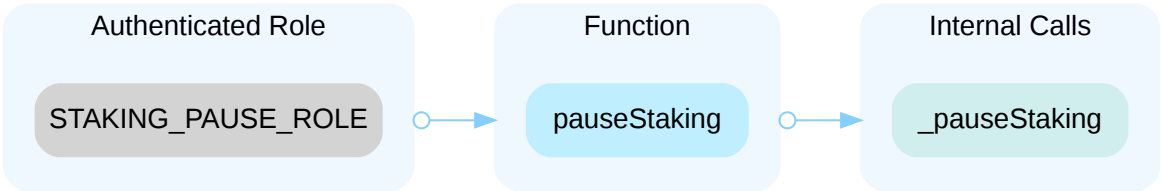

In the contract `StXSAT`, the role `RESUME_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `RESUME_ROLE` account may allow the hacker to take advantage of this authority and resume most of the contract functions.



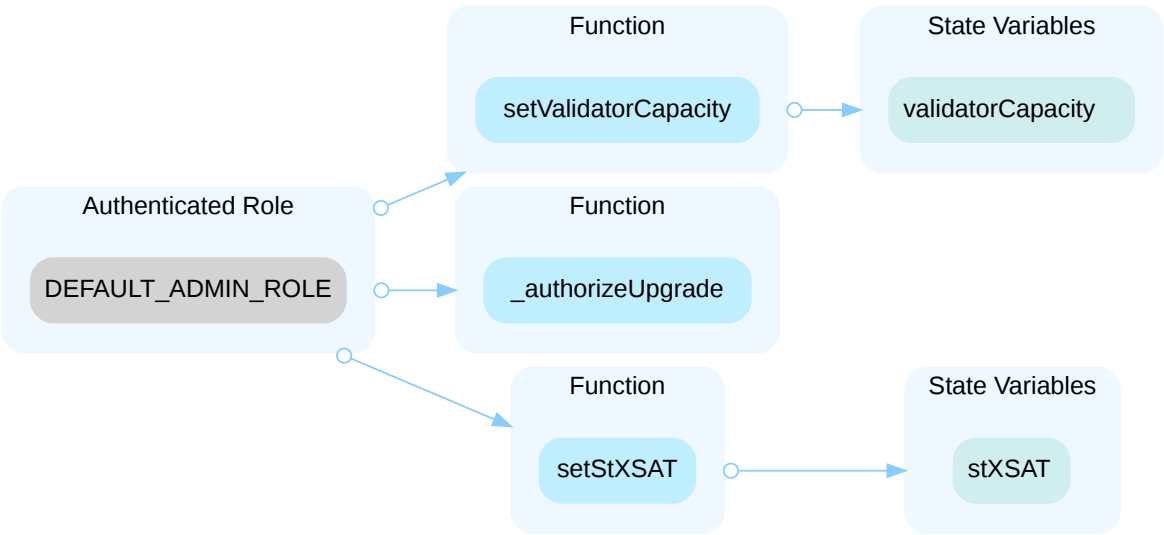

In the contract `StXSAT`, the role `STAKING_CONTROL_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `STAKING_CONTROL_ROLE` account may allow the hacker to take advantage of this authority and resume the staking process or set the staking limit.



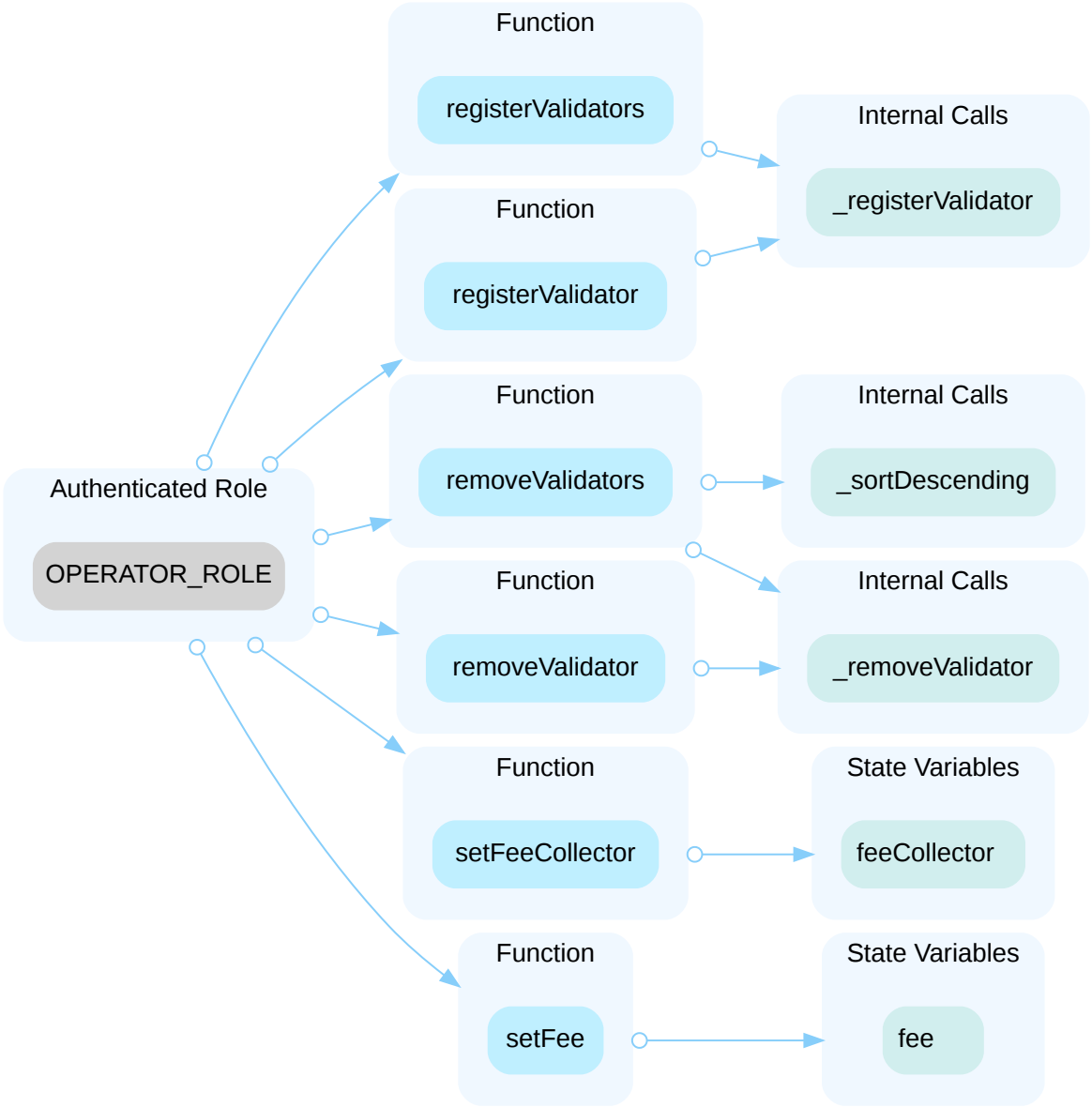

In the contract `StXSAT`, the role `STAKING_PAUSE_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `STAKING_PAUSE_ROLE` account may allow the hacker to take advantage of this authority and pause the staking process.




In the contract `XSATStakingRouter`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and set the validator capacity, authorize new upgrades for the implementation, and set the stXSAT address.




In the contract `XSATStakingRouter`, the role `, OPERATOR_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `OPERATOR_ROLE` account may allow the hacker to take advantage of this authority and register a validator, remove specified validators by given indexes, register multiple validators, set the fee collector address, set the transaction fee, and remove a validator.




The `StXSAT` and `XSATStakingRouter` contracts both inherit from `AccessControlUpgradeable`. The `DEFAULT_ADMIN_ROLE` has authority over the following functions:

- `grantRole`
- `revokeRole`

This means any account assigned the `DEFAULT_ADMIN_ROLE` can manage permissions across the contract by granting or revoking roles to other accounts.

In addition, the `OPERATOR_ROLE` can call `withdrawBTC()` to send the remaining BTC in the contract to the `feeCollector`.

## Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend

centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

### Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;\

AND

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;\

AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

### Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;\

AND

- Introduction of a DAO/governance/voting module to increase transparency and user involvement.\

AND

- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

### Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.\

OR

- Remove the risky functionality.

## Alleviation

**[KiKiFinance, 04/18/2025]**: Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement.

**[CertiK, 04/18/2025]**: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

# KIS-07 | `initialize()` IS UNPROTECTED

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | Medium | contracts/StXSAT.sol (stxsat): 113~114; contracts/XSATStakingRouter.sol (stxsat): 83 | Resolved |

## Description

The `StXSAT` and `XSATStakingRouter` logic contracts do not protect the initializer functions. An attacker can front-run the `initialize` call and assume ownership of the logic contract. Once in control, the attacker can perform privileged operations, misleading users into believing that they are interacting with the legitimate owner of the upgradeable contract.

## Recommendation

We recommend adding

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() initializer {}
```

The addition will prevent the function `initialize()` from being called directly in the implementation contract, but the proxy will still be able to `initialize()` its storage variables.

## Alleviation

**[KiKiFinance, 04/18/2025]**: The team heeded the advice and resolved the issue by adding `constructor` in commit 1beb2441b01cc8897ca72d6f985c563e12423401.

# KIS-09 FUNCTIONS USING `_stakeHelper.deposit()` ARE NON-PAYABLE

| Category | Severity | Location | Status |
|---|---|---|---|
| Denial of Service | Medium | contracts/StXSAT.sol (stxsat): 338; contracts/XSATStakingRouter.sol (stxsat): 257 | Resolved |

## Description

Based on the `IXSATStakeHelper` interface, the `deposit` function is marked as `payable`, indicating that it might require a payment in native tokens via `msg.value`:

```
interface IXSATStakeHelper {
        ...
@>      function deposit(address _target, uint256 _amount) external payable;
```

However, in the call chain `StXSAT.syncBuffers()` → `XSATStakingRouter.deposit()` → `_stakeHelper().deposit()`, the `syncBuffers()` and `deposit()` functions do not include the `payable` keyword:

**`StXSAT.sol`**

```
    function syncBuffers() external nonReentrant whenNotPaused {
        ...
@>          router.deposit(amountToDeposit);
            emit TokenDepositSynced(stakingRouter, amountToDeposit);
        }
```

**`XSATStakingRouter.sol`**

```
    function deposit(uint256 _amount) external
onlyStXSAT nonReentrant whenNotPaused {
            ...
            // Stake a fixed amount (validatorCapacity) to the found validator.
@>          _stakeHelper().deposit(validators[usedIndex].validatorAddress,
validatorCapacity);
```

As a result, these function calls do not carry any `msg.value`, potentially causing `_stakeHelper().deposit()` to revert due to insufficient native tokens.

For example, if the `_stakeHelper()` contract is the `StakeHelper.sol` deployed at 0xF80172f40b0C964c9e1787A06950268EF4BE48B3, it requires BTC as a fee. If `depositFee` is non-zero, the `deposit()` function will revert, causing all functions in the call chain to also revert.

```
    function deposit(address _target, uint256 _amount) public payable {
        ...
@>      require(msg.value == depositFee,
"Deposit: must pay exact amount of deposit fee");
```

## Recommendation

It is recommended to make the `StXSAT.syncBuffers()` and `XSATStakingRouter.deposit()` functions `payable` to ensure they can handle the required native token payment.

## Alleviation

**[KiKiFinance, 04/18/2025]**: The team heeded the advice and resolved the issue by adding `payable` keywords in commit c175e7497441d1b575b50ddf4fc7519f7d18df80.

# KIS-23 POTENTIAL XSAT TOKEN FLOW DISRUPTION DUE TO DELAYED `syncBuffers` EXECUTION AND VALIDATOR CAPACITY MISMATCH

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | Medium | contracts/StXSAT.sol (stxsat): 322~323 | Acknowledged |

## Description

The `syncBuffers` function is responsible for managing the XSAT token flow between the `stXSAT` contract and the `router` based on changes in `_depositBuffer` and `_withdrawalReserve`. Specifically, it can trigger deposits or withdrawals by calling the router's `deposit` and `requestWithdraw` functions, respectively.

However, there are several considerations and potential issues related to these operations:

### 1. Execution Timing and Buffer Accumulation

- If `syncBuffers()` is not called for an extended period, the `_depositBuffer` may accumulate a large amount of XSAT tokens.
- This can lead to an excessively large `depositUnits` and `amountToDeposit`, which are passed to `router.deposit()`. Inside `router.deposit()`, the following logic is used:

```
uint256 count = _amount / validatorCapacity;
...
require(found, "All validators are staked");
```

- If `count` exceeds the number of available (unstaked) validators, the loop cannot find a validator to stake, causing the transaction to revert with `"All validators are staked"`.

### 2. Potential Reverts in Deposit and Withdrawal

- The `validatorCapacity` value, which defines how much XSAT each validator handles, can be updated by the `admin` through `setValidatorCapacity`.
- If `validatorCapacity` is set to a **small** value, the resulting `count` (calculated as `_amount / validatorCapacity`), becomes large. This increases the risk of running out of available validators, leading to reverts in both the `router.deposit()` and `router.requestWithdraw()` functions due to insufficient unstaked or staked validators, respectively.
- If `syncBuffers()` reverts, then the `stXSAT` contract **may not receive sufficient XSAT tokens** from the router. As a result, **users may be unable to redeem their stXSAT for XSAT**, leading to **user funds being stuck** and potential

financial loss.

### 3. Validator Capacity Update Risks

- In the `router.deposit()` function, each validator's staked amount is set based on the current `validatorCapacity`:

```
validators[usedIndex].amount = validatorCapacity;
```

- If `validatorCapacity` is **updated after deposits are made but before a withdrawal is triggered**, several issues may arise:
    - The stored `amount` for each validator reflects the **old** `validatorCapacity`, creating a mismatch between the actual staked amount and the expected capacity during withdrawal.
    - When `syncBuffers()` executes, it uses the **updated** `validatorCapacity` to calculate the withdrawal amount:

```
uint256 withdrawalUnits = deficit / validatorCapacity;
...
uint256 amountToWithdraw = withdrawalUnits * validatorCapacity;
```

    - However, inside `router.requestWithdraw()`, the logic sets each validator's amount to zero without considering whether the `validatorCapacity` used during deposit matches the updated value:

```
@>    _stakeHelper().withdraw(validators[usedIndex].validatorAddress,
validatorCapacity);
      validators[usedIndex].staked = false;
@>    validators[usedIndex].amount = 0;
```

    - `_stakeHelper()` is a third-party dependency, and its `deposit()` and `withdraw()` implementations are not available for review. As such, it is difficult to guarantee that setting `validators[usedIndex].amount` to 0 immediately after calling `withdraw()` is correct or aligns with the intended integration logic.
    - If `_stakeHelper()` relies on the recorded amount for validator state tracking or validation, this operation could result in inconsistencies, unexpected reverts, or even financial loss.

## Recommendation

We recommend the team to:

- Ensure that both the `router` and `_stakeHelper` contracts maintain sufficient XSAT token balances at all times to prevent user funds from becoming locked.
- Carefully manage all state variable updates to avoid interrupting user operations or causing potential fund lockups.
- Conduct a thorough review of the third-party contract logic and verify that there are no inconsistencies between state updates in the current contracts and the behavior of the integrated third-party components.

## Alleviation

**[KiKiFinance, 04/19/2025]**: Issue acknowledged. I won't make any changes for the current version. ValidatorCapacity is the fixed staking amount for exSat network validators; it generally wont change, and if it does, we will upgrade the contract.

**[CertiK, 04/21/2025]**: The team has updated the `syncBuffers()` function to allow the caller to specify a `_limitUnit` parameter in commit 4b3d07c13ddd415f3b30bd6fa8c0f2f28c007fa1, which helps prevent the calculated `count` from becoming too large and causing the function to revert.

```
        function syncBuffers(uint256 _limitUnit) external payable nonReentrant
whenNotPaused {
```

However, the effectiveness of this safeguard heavily depends on the `_limitUnit` value provided by the caller. If the input is not set properly, the function may still revert.

Additionally, it is recommended to ensure that `_limitUnit` is greater than zero. If not, the calculated `depositUnits` will be set to zero:

```
        require(depositUnits > 0, "Deposit buffer insufficient for one unit
deposit");
            if (depositUnits > _limitUnit) {
                depositUnits = _limitUnit;
            }
```

# KIS-24 | USING `msg.value` INSIDE A LOOP

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Issue | Medium | contracts/XSATStakingRouter.sol (0418-1beb24): 275~276 | Resolved |

## Description

`msg.value` is used inside a loop. The incoming payment may be processed multiple times unexpectedly.

```
275             _stakeHelper().deposit{value: msg.value}(validators[usedIndex].
validatorAddress, validatorCapacity);
```

## Recommendation

We recommend tracking the remaining amount using a local variable and decrease its amount on each iteration/usage.

## Alleviation

**[KiKiFinance, 04/21/2025]**: The team heeded the advice and resolved the issue, the change is reflected in commit `d0569f30d2311c94bb31fa0ff8da23169e74b112`.

**[CertiK, 04/21/2025]**: The team has added a check to ensure that the `msg.value` matches the expected total deposit fee, calculated as `count * depositFee`, for each deposit made through the `_stakeHelper` contract.

```
require(msg.value == depositFee * count, "Incorrect total deposit fee");
```

Please note that this check only functions correctly when `count` is less than or equal to the number of available unstaked validators. If `count` exceeds the number of available validators, the function will revert. This behavior is discussed in detail in issue KIS-23.

# KIS-08 | LACK OF STORAGE GAP IN UPGRADEABLE CONTRACT

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | Minor | contracts/BaseStXSAT.sol (stxsat): 10 | Resolved |

## Description

There is no storage gap preserved in the logic contract. Any logic contract that acts as a base contract that needs to be inherited by other upgradeable child should have a reasonable size of storage gap preserved for the new state variable introduced by the future upgrades.

## Recommendation

We recommend having a storage gap of a reasonable size preserved in the logic contract in case that new state variables are introduced in future upgrades. For more information, please refer to: https://docs.openzeppelin.com/contracts/3.x/upgradeable#storage_gaps.

## Alleviation

**[KiKiFinance, 04/18/2025]**: The team heeded the advice and resolved the issue by adding a storage gap in the base contract in commit 4c6647bab39284af174ea0c23e875b130e54572c.

# KIS-10 | NO UPPER LIMIT IN `setFee()` FUNCTION

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | Minor | contracts/XSATStakingRouter.sol (stxsat): 349 | Acknowledged |

## Description

There are no upper boundaries for `_fee` which is used in `setFee()`. It is possible to set the fee rate up to any arbitrary amount. In this case, the `StSAT` token's reward may incur a fee exceeding 100%, resulting in excessive shares being minted to the `feeCollector` during `_distributeFee`.

## Recommendation

We recommend adding reasonable boundaries for the fees.

## Alleviation

**[KiKiFinance, 04/18/2025]**: Issue acknowledged. I won't make any changes for the current version. Because the operating costs of the xSAT validator are uncertain it currently covers BTC gas fees and server expenses and we cannot assess the upper limit of these costs at this time, we will not set an upper bound for now.

# KIS-11 | `safeApprove()` IS DEPRECATED

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | Minor | contracts/StXSAT.sol (stxsat): 336; contracts/XSATStakingRouter.sol (stxsat): 255 | Acknowledged |

## Description

`safeApprove()` has been deprecated in favor of `safeIncreaseAllowance()` and `safeDecreaseAllowance()`. If only setting the initial allowance to the value that means infinite, `safeIncreaseAllowance()` can be used instead. The function may currently work, but if a bug is found in this version of OpenZeppelin, and the recommended version no longer has this function, there will be unnecessary delays in porting and testing replacement contracts.

## Recommendation

We recommend using `safeIncreaseAllowance()` and `safeDecreaseAllowance()` instead.

## Alleviation

**[KiKiFinance, 04/18/2025]**: Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement.

# KIS-12 | MISSING ZERO ADDRESS VALIDATION

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | Minor | contracts/StXSAT.sol (stxsat): 120~121; contracts/XSATStakingRouter.sol (stxsat): 96 | Resolved |

## Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

```
120         xsatToken = _xsatAddress;
121         stakingRouter = _stakeRouterAddress;
```

- `_xsatAddress` and `_stakeRouterAddress` are not zero-checked before being used.

```
95         rewardHelper = _rewardHelperAddress;
```

- `_rewardHelperAddress` are not zero-checked before being used.

## Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

## Alleviation

**[KiKiFinance, 04/18/2025]**: The team heeded the advice and resolved the issue by adding zero-checks for the input addresses in commit 7f0803dd3b4c09c8fb41c67a53632364c82392fe.

# KIS-13 | MISSING EMIT EVENTS

| Category | Severity | Location | Status |
|---|---|---|---|
| Inconsistency | Minor | contracts/XSATStakingRouter.sol (stxsat): 104~107 | Resolved |

## Description

The smart contract contains one or more state changes related to access control that do not emit events to communicate the changes outside the blockchain, which can lead to difficulties in tracking or verifying these changes and may affect the contract's transparency and auditability.

```
106         stXSAT = _stXSAT;
```

## Recommendation

It is suggested to declare and emit corresponding events for all the essential state variables that are possible to be changed during runtime.

## Alleviation

**[KiKiFinance, 04/18/2025]**: The team heeded the advice and resolved the issue by emitting an event in commit 778b119f4cb421be5a170f79493492ae6fbedbfe.

# KIS-14 | EXTERNAL CALL INSIDE LOOP

| Category | Severity | Location | Status |
|---|---|---|---|
| Denial of Service | Minor | contracts/XSATStakingRouter.sol (stxsat): 233~264, 271~294, 316~325 | Acknowledged |

## Description

External calls are made inside a `for` loop. This might lead to a denial-of-service attack. If any of the calls fail, it will cause the entire loop to revert.

```
322             _rewardHelper().vdrclaim(validator.validatorAddress);
```

```
257             _stakeHelper().deposit(validators[usedIndex].validatorAddress,
validatorCapacity);
```

```
288             _stakeHelper().withdraw(validators[usedIndex].validatorAddress,
validatorCapacity);
```

## Recommendation

It is advised to refactor the code to move external calls outside the loop, or use alternative patterns such as the "Withdrawal Pattern" to minimize the risk of denial-of-service attacks and improve the overall security of the smart contract.

## Alleviation

**[KiKiFinance, 04/18/2025]**: Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement.

# KIS-15 | POTENTIAL REENTRANCY ATTACK

| Category | Severity | Location | Status |
|---|---|---|---|
| Concurrency | Minor | contracts/XSATStakingRouter.sol (stxsat): 316~325 | Resolved |

## Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

### External call(s)

```
322            _rewardHelper().vdrclaim(validator.validatorAddress);
```

## Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

## Alleviation

**[KiKiFinance, 04/18/2025]**: The team heeded the advice and resolved the issue by adding `nonReentrant` modifier in commit e2ff04b0810ee1de86b11231fe77a9253278029f.

# KIS-16 | MISSING INPUT VALIDATION

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | Minor | contracts/StXSAT.sol (stxsat): 173~174 | Resolved |

## Description

In the `StXSAT` contract, the `setStakingLimit` function allows the `stakingController` to set the maximum staking limit for the pool. However, there is no validation on the `_maxStakeLimit` parameter. The `maxStakeLimit` is utilized in the `_getCurrentStakeLimit` function as follows:

```
        return maxStakeLimit - _stakingRouter().getStakingBalance() -
_depositBuffer;
```

If the `maxStakeLimit` is set arbitrarily, users will be unable to deposit because `_getCurrentStakeLimit` will always revert due to an underflow.

## Recommendation

It is recommended to add input validations in the aforementioned functions to prevent potential errors.

## Alleviation

**[KiKiFinance, 04/18/2025]**: The team heeded the advice and resolved the issue by adding input validation in commit afd82ed16674652961be332893ba03560a5d6dd5.

# KIS-20 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | Minor | contracts/StXSAT.sol (stxsat): 372; contracts/XSATStakingRouter.sol (stxsat): 239~240 | Resolved |

## Description

The project design may not be compatible with non-standard ERC20 tokens, such as deflationary tokens.

The functions use `safeTransferFrom()` to move funds from the sender to the recipient but fail to verify if the received token amount matches the transferred amount. This could pose an issue with fee-on-transfer tokens, where the post-transfer balance might be less than anticipated, leading to balance inconsistencies. There might be subsequent checks for a second transfer, but an attacker might exploit leftover funds (such as those accidentally sent by another user) to gain unjustified credit.

## Scenario

When transferring deflationary ERC20 tokens, the input amount may not equal the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrive to the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

## Recommendation

We advise the client to regulate the set of tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support non-standard ERC20 tokens.

## Alleviation

**[KiKiFinance, 04/18/2025]**: The team heeded the advice and resolved the issue by adding balance tracking mechanism in commit 2d5bcdbf20df54169e336e547a963f89a73b5e64.

# KIS-22 `removeValidators()` SHOULD CHECK FOR DUPLICATE ADJACENT INDEXES

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | Minor | contracts/XSATStakingRouter.sol (stxsat): 179 | Resolved |

## Description

After performing selection sort in descending order, `removeValidators` should check whether adjacent indexes in the `sortedIndexes` array are the same, to avoid deleting the same index twice and accidentally removing other indexes.

## Recommendation

It is recommended that a duplicate index check be added as described in the Description.

## Alleviation

**[KiKiFinance, 04/18/2025]**: The team heeded the advice and resolved the issue by adding a duplicate index check in commit 5ba8834ca90408b5d0912bd0c51b078f054ff2d5.

# KIS-17 | UNINITIALIZED LOCAL VARIABLE

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Issue | Informational | contracts/XSATStakingRouter.sol (stxsat): 243, 277 | Resolved |

## Description

The smart contract contains one or more uninitialized local variables, which can reduce readability and may introduce vulnerabilities.

```
277             uint256 usedIndex;
```

- Local variable `usedIndex` is never initialized.

```
243             uint256 usedIndex;
```

- Local variable `usedIndex` is never initialized.

## Recommendation

It is advised to initialize all local variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability and avoid potential issues.

## Alleviation

**[KiKiFinance, 04/18/2025]**: The team heeded the advice and resolved the issue by initializing the local variables in commit 15f9e50e1882700e97e52f8cbc649a0ec171fd7a.

# KIS-19 INCONSISTENCY IN REWARD AND FEE LOGIC IN SOLIDITY IMPLEMENTATION

| Category | Severity | Location | Status |
|---|---|---|---|
| Inconsistency | Informational | contracts/StXSAT.sol (stxsat): 511~512 | Resolved |

## Description

In the tech report, step C outlines the calculation of reward shares, followed by step D, which involves calling the `_distributeFee` function to mint shares to the fee collector. However, there is an inconsistency in the code implementation. The current code distributes fees and mints shares before calculating the reward shares. This will result in `rewardShare` being lower than expected, because `_totalShares` was increased beforehand in `_distributeFee`. Additionally, the local variable `rewardShares` is declared but not used within the function, nor is it included in any event.

```
function _completeTokenRebase(uint256 _reward) internal returns (uint256 feeShares)
{
        ...

        // Distribute fee and retrieve the fee shares minted.
        feeShares = _distributeFee(preTotalPooledXSAT, preTotalShares, _reward);

        // Calculate new pooled XSAT.
        uint256 postTotalPooledXSAT = preTotalPooledXSAT + _reward;

        // Calculate shares corresponding to the reward.
@>      uint256 rewardShares = getSharesByPooledXSAT(_reward);

        ...
        emit TokenRebased(
            preTotalShares,
            preTotalPooledXSAT,
            postTotalShares,
            postTotalPooledXSAT,
            feeShares
        );
}
```

## Recommendation

We recommend confirming with the team to ensure consistency between the tech report and the code implementation. It is also advisable to include the `rewardShares` variable in the emitted event for transparency.

## Alleviation

**[KiKiFinance, 04/18/2025]**: The team heeded the advice and resolved the issue by removing the `rewardShares` variable in commit eb0dcf5e3070ab1e5955969c51acb831340238df.

# KIS-21 | DISCUSSION ON `stXSAT` SETTING

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | Informational | contracts/XSATStakingRouter.sol (stxsat): 106~107 | Resolved |

## Description

In the `XSATStakingRouter` contract, the `stXSAT` contract is authorized to call these functions:

- `deposit`
- `requestWithdraw`
- `collectRewards`

Within the `StXSAT` contract, the following functions interact with the router contract:

- `collectRewards` invokes `router.collectRewards()`
- `syncBuffers` invokes both `router.deposit()` and `router.requestWithdraw()`

However, the `admin` can update the `stXSAT` address at any time. If it's updated to a new contract, inconsistent states may arise between the old `stXSAT` and the router contract. If users continue interacting with the old `stXSAT` contract without being aware of the change, they won't receive rewards from it.

## Recommendation

We would like to discuss this finding with the team to ensure it aligns with the intended design.

## Alleviation

**[KiKiFinance, 04/18/2025]**: The team heeded the advice and resolved the issue by restricting the `stXSAT` setting in commit 9bd3c4834d245bb60000941bfafdd07efa528872.

# OPTIMIZATIONS | KIKIFINANCE - STXSAT

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| KIS-01 | `For` Or `While` Loops May Consume Excessive Gas And Even Lead To DoS | Gas Optimization | Optimization | Acknowledged |
| KIS-02 | Use Named Return Values | Logical Issue | Optimization | Acknowledged |
| KIS-03 | State Variables That Could Be Declared Immutable | Coding Issue | Optimization | Acknowledged |

# KIS-01 `For` OR `While` LOOPS MAY CONSUME EXCESSIVE GAS AND EVEN LEAD TO DOS

| Category | Severity | Location | Status |
|---|---|---|---|
| Gas Optimization | Optimization | contracts/StXSAT.sol (stxsat): 232~241; contracts/XSATStakingRouter.sol (stxsat): 152, 241~263, 245~252, 275~293, 279~286, 375~377 | Acknowledged |

## Description

Using `for` or `while` loops in the contract can consume a significant amount of gas. When nested loops are used, this gas consumption is further amplified and may even lead to a denial-of-service (DoS).

## Recommendation

We recommend reducing the number of loop iterations or redesigning to avoid loop structures in order to lower gas consumption.

## Alleviation

**[KiKiFinance, 04/18/2025]**: Issue acknowledged. I won't make any changes for the current version.

# KIS-02 | USE NAMED RETURN VALUES

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | Optimization | contracts/BaseStXSAT.sol (stxsat): 75, 79, 84, 88, 93, 99, 104, 120, 125, 130, 212; contracts/StXSAT.sol (stxsat): 192, 368; contracts/XSATStakingRouter.sol (stxsat): 151; contracts/wstXSAT.sol (stxsat): 90, 99 | Acknowledged |

## Description

Using named return values instead of explicitly calling `return` saves ~13 execution gas per call and >1000 deployment gas per instance.

## Recommendation

We recommend using named return values.

## Alleviation

**[KiKiFinance, 04/18/2025]**: The team acknowledged the issue and decided to remain unchanged in the scope of the audit.

# KIS-03 STATE VARIABLES THAT COULD BE DECLARED IMMUTABLE

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Issue | Optimization | contracts/wstXSAT.sol (stxsat): 27 | Acknowledged |

## Description

State variables that are not updated following deployment should be declared immutable to save gas.

## Recommendation

Add the immutable attribute to state variables that never change or are set only in the constructor.

## Alleviation

**[KiKiFinance, 04/18/2025]**: The team acknowledged the issue and decided to remain unchanged in the scope of the audit.

# FORMAL VERIFICATION | KIKIFINANCE - STXSAT

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

## Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

### Verification of contracts derived from AccessControl v4.4

We verified properties of the public interface of contracts that provide an AccessControl-v4.4 compatible API. This involves:

- The `hasRole` function, which returns `true` if an account has been granted a specific `role`.
- The `getRoleAdmin` function, which returns the admin role that controls a specific `role`.
- The `grantRole` and `revokeRole` functions, which are used for granting a `role` to an account and revoking a `role` from an `account`, respectively.
- The `renounceRole` function, which allows the calling account to revoke a `role` from itself.

The properties that were considered within the scope of this audit are as follows:

| Property Name | Title |
|---|---|
| accesscontrol-getroleadmin-change-state | `getRoleAdmin` Function Does Not Change State |
| accesscontrol-hasrole-change-state | `hasRole` Function Does Not Change State |
| accesscontrol-getroleadmin-succeed-always | `getRoleAdmin` Function Always Succeeds |
| accesscontrol-hasrole-succeed-always | `hasRole` Function Always Succeeds |
| accesscontrol-renouncerole-revert-not-sender | `renounceRole` Reverts When Caller Is Not the Confirmation Address |
| accesscontrol-default-admin-role | AccessControl Default Admin Role Invariance |
| accesscontrol-renouncerole-succeed-role-renouncing | `renounceRole` Successfully Renounces Role |
| accesscontrol-revokerole-correct-role-revoking | `revokeRole` Correctly Revokes Role |
| accesscontrol-grantrole-correct-role-granting | `grantRole` Correctly Grants Role |

### Verification of ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows (note that overflow properties were excluded from the verification):

| Property Name | Title |
|---|---|
| erc20-transfer-exceed-balance | `transfer` Fails if Requested Amount Exceeds Available Balance |
| erc20-transfer-never-return-false | `transfer` Never Returns `false` |
| erc20-allowance-change-state | `allowance` Does Not Change the Contract's State |
| erc20-transferfrom-fail-exceed-balance | `transferFrom` Fails if the Requested Amount Exceeds the Available Balance |
| erc20-transferfrom-fail-exceed-allowance | `transferFrom` Fails if the Requested Amount Exceeds the Available Allowance |
| erc20-transferfrom-correct-amount | `transferFrom` Transfers the Correct Amount in Transfers |
| erc20-transfer-correct-amount | `transfer` Transfers the Correct Amount in Transfers |
| erc20-transferfrom-correct-allowance | `transferFrom` Updated the Allowance Correctly |
| erc20-approve-succeed-normal | `approve` Succeeds for Valid Inputs |
| erc20-approve-never-return-false | `approve` Never Returns `false` |
| erc20-allowance-correct-value | `allowance` Returns Correct Value |
| erc20-approve-correct-amount | `approve` Updates the Approval Mapping Correctly |
| erc20-balanceof-correct-value | `balanceOf` Returns the Correct Value |
| erc20-approve-false | If `approve` Returns `false`, the Contract's State Is Unchanged |
| erc20-transfer-false | If `transfer` Returns `false`, the Contract State Is Not Changed |
| erc20-totalsupply-succeed-always | `totalSupply` Always Succeeds |

| Property Name | Title |
|---|---|
| erc20-totalsupply-correct-value | `totalSupply` Returns the Value of the Corresponding State Variable |
| erc20-balanceof-succeed-always | `balanceOf` Always Succeeds |
| erc20-transferfrom-false | If `transferFrom` Returns `false`, the Contract's State Is Unchanged |
| erc20-balanceof-change-state | `balanceOf` Does Not Change the Contract's State |
| erc20-approve-revert-zero | `approve` Prevents Approvals For the Zero Address |
| erc20-transferfrom-revert-zero-argument | `transferFrom` Fails for Transfers with Zero Address Arguments |
| erc20-totalsupply-change-state | `totalSupply` Does Not Change the Contract's State |
| erc20-transfer-revert-zero | `transfer` Prevents Transfers to the Zero Address |
| erc20-transferfrom-never-return-false | `transferFrom` Never Returns `false` |
| erc20-allowance-succeed-always | `allowance` Always Succeeds |

## Verification Results

For the following contracts, formal verification established that each of the properties that were in scope of this audit (see scope) are valid:

### Detailed Results For Contract XSATStakingRouter (contracts/XSATStakingRouter.sol) In Commit 34f529e9e4513bbb0f40e2348bfb60ba88c6cc2e

#### Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `getRoleAdmin`

| Property Name | Final Result | Remarks |
|---|---|---|
| accesscontrol-getroleadmin-change-state | ● True | |
| accesscontrol-getroleadmin-succeed-always | ● True | |

Detailed Results for Function `hasRole`

| Property Name | Final Result | Remarks |
|---|---|---|
| accesscontrol-hasrole-change-state | ● True | |
| accesscontrol-hasrole-succeed-always | ● True | |

Detailed Results for Function `renounceRole`

| Property Name | Final Result | Remarks |
|---|---|---|
| accesscontrol-renouncerole-revert-not-sender | ● True | |
| accesscontrol-renouncerole-succeed-role-renouncing | ● True | |

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

| Property Name | Final Result | Remarks |
|---|---|---|
| accesscontrol-default-admin-role | ● True | |

Detailed Results for Function `grantRole`

| Property Name | Final Result | Remarks |
|---|---|---|
| accesscontrol-grantrole-correct-role-granting | ● True | |

Detailed Results for Function `revokeRole`

| Property Name | Final Result | Remarks |
|---|---|---|
| accesscontrol-revokerole-correct-role-revoking | ● True | |

## Detailed Results For Contract StXSAT (contracts/StXSAT.sol) In Commit 34f529e9e4513bbb0f40e2348bfb60ba88c6cc2e

### Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `getRoleAdmin`

| Property Name | Final Result | Remarks |
|---|---|---|
| accesscontrol-getroleadmin-succeed-always | ● True | |
| accesscontrol-getroleadmin-change-state | ● True | |

Detailed Results for Function `hasRole`

| Property Name | Final Result | Remarks |
|---|---|---|
| accesscontrol-hasrole-succeed-always | ● True | |
| accesscontrol-hasrole-change-state | ● True | |

Detailed Results for Function `renounceRole`

| Property Name | Final Result | Remarks |
|---|---|---|
| accesscontrol-renouncerole-revert-not-sender | ● True | |
| accesscontrol-renouncerole-succeed-role-renouncing | ● True | |

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

| Property Name | Final Result | Remarks |
|---|---|---|
| accesscontrol-default-admin-role | ● True | |

Detailed Results for Function `revokeRole`

| Property Name | Final Result | Remarks |
|---|---|---|
| accesscontrol-revokerole-correct-role-revoking | ● True | |

Detailed Results for Function `grantRole`

| Property Name | Final Result | Remarks |
|---|---|---|
| accesscontrol-grantrole-correct-role-granting | ● True | |

### Detailed Results For Contract WstXSAT (contracts/wstXSAT.sol) In Commit 34f529e9e4513bbb0f40e2348bfb60ba88c6cc2e

**Verification of ERC-20 Compliance**

Detailed Results for Function `transfer`

| Property Name | Final Result | Remarks |
|---|---|---|
| erc20-transfer-exceed-balance | ● True | |
| erc20-transfer-never-return-false | ● True | |
| erc20-transfer-correct-amount | ● True | |
| erc20-transfer-false | ● True | |
| erc20-transfer-revert-zero | ● True | |

Detailed Results for Function `allowance`

| Property Name | Final Result | Remarks |
|---|---|---|
| erc20-allowance-change-state | ● True | |
| erc20-allowance-correct-value | ● True | |
| erc20-allowance-succeed-always | ● True | |

Detailed Results for Function `transferFrom`

| Property Name | Final Result | Remarks |
|---|---|---|
| erc20-transferfrom-fail-exceed-balance | ● True | |
| erc20-transferfrom-fail-exceed-allowance | ● True | |
| erc20-transferfrom-correct-amount | ● True | |
| erc20-transferfrom-correct-allowance | ● True | |
| erc20-transferfrom-false | ● True | |
| erc20-transferfrom-revert-zero-argument | ● True | |
| erc20-transferfrom-never-return-false | ● True | |

Detailed Results for Function `approve`

| Property Name | Final Result | Remarks |
|---|---|---|
| erc20-approve-succeed-normal | ● True | |
| erc20-approve-never-return-false | ● True | |
| erc20-approve-correct-amount | ● True | |
| erc20-approve-false | ● True | |
| erc20-approve-revert-zero | ● True | |

Detailed Results for Function `balanceOf`

| Property Name | Final Result | Remarks |
|---|---|---|
| erc20-balanceof-correct-value | ● True | |
| erc20-balanceof-succeed-always | ● True | |
| erc20-balanceof-change-state | ● True | |

Detailed Results for Function `totalSupply`

| Property Name | Final Result | Remarks |
|---|---|---|
| erc20-totalsupply-succeed-always | ● True | |
| erc20-totalsupply-correct-value | ● True | |
| erc20-totalsupply-change-state | ● True | |

# APPENDIX | KIKIFINANCE - STXSAT

## Finding Categories

| Categories | Description |
|---|---|
| Gas Optimization | Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction. |
| Coding Issue | Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues. |
| Denial of Service | Denial of Service findings indicate that an attacker may prevent the program from operating correctly or responding to legitimate requests. |
| Concurrency | Concurrency findings are about issues that cause unexpected or unsafe interleaving of code executions. |
| Inconsistency | Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification. |
| Volatile Code | Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities. |
| Logical Issue | Logical Issue findings indicate general implementation issues related to the program logic. |
| Centralization | Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code. |

## Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

## Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

## Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

## Description of the Analyzed AccessControl-v4.4 Properties

### Properties related to function `getRoleAdmin`

#### accesscontrol-getroleadmin-change-state

The `getRoleAdmin` function must not change any state variables.

Specification:

```
assignable \nothing;
```

#### accesscontrol-getroleadmin-succeed-always

The `getRoleAdmin` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

**Properties related to function `hasRole`**

**accesscontrol-hasrole-change-state**

The `hasRole` function must not change any state variables.

Specification:

```
assignable \nothing;
```

**accesscontrol-hasrole-succeed-always**

The `hasRole` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

**Properties related to function `renounceRole`**

**accesscontrol-renouncerole-revert-not-sender**

The `renounceRole` function must revert if the caller is not the same as `account`.

Specification:

```
reverts_when account != msg.sender;
```

**accesscontrol-renouncerole-succeed-role-renouncing**

After execution, `renounceRole` must ensure the caller no longer has the renounced role.

Specification:

```
ensures !hasRole(role, account);
```

**Properties related to function `DEFAULT_ADMIN_ROLE`**

**accesscontrol-default-admin-role**

The default admin role must be invariant, ensuring consistent access control management.

Specification:

```
invariant DEFAULT_ADMIN_ROLE() == 0x00;
```

#### Properties related to function `revokeRole`

##### accesscontrol-revokerole-correct-role-revoking

After execution, `revokeRole` must ensure the specified account no longer has the revoked role.

Specification:

```
ensures !hasRole(role, account);
```

#### Properties related to function `grantRole`

##### accesscontrol-grantrole-correct-role-granting

After execution, `grantRole` must ensure the specified account has the granted role.

Specification:

```
ensures hasRole(role, account);
```

### Description of the Analyzed ERC-20 Properties

#### Properties related to function `transfer`

##### erc20-transfer-correct-amount

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

Specification:

```
requires recipient != msg.sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(recipient) == \old(balanceOf(recipient) + amount)
&& balanceOf(msg.sender) == \old(balanceOf(msg.sender) - amount);
  also
requires recipient == msg.sender;
ensures \result ==> balanceOf(msg.sender) == \old(balanceOf(msg.sender));
```

##### erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

Specification:

```
requires amount > balanceOf(msg.sender);
ensures !\result;
```

#### erc20-transfer-false

If the `transfer` function in contract `WstXSAT` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

#### erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

Specification:

```
ensures \result;
```

#### erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

Specification:

```
ensures \old(recipient) == address(0) ==> !\result;
```

### Properties related to function `allowance`

#### erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

#### erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

Specification:

```
ensures \result == allowance(\old(owner), \old(spender));
```

#### erc20-allowance-succeed-always

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

### Properties related to function `transferFrom`

#### erc20-transferfrom-correct-allowance

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount`.

Specification:

```
ensures \result ==> allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) - \old(amount)
                  || (allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) && \old(allowance(sender, msg.sender)) == type(uint256).max);
```

#### erc20-transferfrom-correct-amount

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest`.

Specification:

```
requires recipient != sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient) +
amount)
                  && balanceOf(\old(sender)) == \old(balanceOf(sender) - amount);
  also
requires recipient == sender;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient));
```

#### erc20-transferfrom-fail-exceed-allowance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

Specification:

```
requires msg.sender != sender;
requires amount > allowance(sender, msg.sender);
ensures !\result;
```

#### erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

Specification:

```
requires amount > balanceOf(sender);
ensures !\result;
```

#### erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

#### erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false`.

Specification:

```
ensures \result;
```

#### erc20-transferfrom-revert-zero-argument

All calls of the form `transferFrom(from, dest, amount)` must fail for transfers from or to the zero address.

Specification:

```
ensures \old(sender) == address(0) ==> !\result;
also
ensures \old(recipient) == address(0) ==> !\result;
```

### Properties related to function `approve`

#### erc20-approve-correct-amount

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance

mapping according to the address `msg.sender` and the values of `spender` and `amount`.

Specification:

```
requires spender != address(0);
ensures \result ==> allowance(msg.sender, \old(spender)) == \old(amount);
```

**erc20-approve-false**

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

**erc20-approve-never-return-false**

The function `approve` must never returns `false`.

Specification:

```
ensures \result;
```

**erc20-approve-revert-zero**

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

Specification:

```
ensures \old(spender) == address(0) ==> !\result;
```

**erc20-approve-succeed-normal**

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

Specification:

```
requires spender != address(0);
ensures \result;
reverts_only_when false;
```

### Properties related to function `balanceOf`

#### erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

#### erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`.

Specification:

```
ensures \result == balanceOf(\old(account));
```

#### erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

### Properties related to function `totalSupply`

#### erc20-totalsupply-change-state

The `totalSupply` function in contract WstXSAT must not change any state variables.

Specification:

```
assignable \nothing;
```

#### erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract WstXSAT.

Specification:

```
ensures \result == totalSupply();
```

#### erc20-totalsupply-succeed-always

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

# DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

# Elevating Your Entire Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.