KREDICT

Smart Contract Audit

SWEDEN (SWED)

Deliverable
Smart Contract Audit Report
Security Report **Apr 2025**

Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Company. The content, conclusions, and recommendations in this publication are elaborated in the specific for only the project.

KREDICT does not guarantee the accuracy of the data included in this study. All representations, warranties, undertakings, and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Company nor any personating on the Company's behalf may be held responsible for the use that may be made of the information contained herein.

KREDICT retains the right to display audit reports and other content elements as examples of their work in their portfolio and as content features in other projects to protect all customer security purposes. The description containing confidential information can be used internally by the Customer or disclosed publicly after all vulnerabilities are fixed - upon the Customer's decision.

Overview

Background

SWEDEN requested that KREDICT perform an Extensive Smart Contract audit of their Smart Contract.

Project Dates

The following is the project schedule for this review and report:

Apr 6, 2025

➔ Smart Contract Review Completed (Completed)

Mar 6, 2025

➔ Delivery of Smart Contract Audit Report (Completed)

Review Team

The following KREDICT team members participated in this review:

➔ **Abhishek Mishra**, Security Researcher, and Engineer

Coverage

Target Specification and Revision

For this audit, we performed research, investigated, and reviewed the smart contract of **SWEDEN**.

The following documentation repositories were considered in-scope for the review:

➔ **SWEDEN (SWED)**

Explorer Link

TLomkn2JKndsKM5c8g8cKnzAiGpGyUUxaR

Token Name	SWEDEN
Symbol	SWED
Chain	Tron
Contract Address	TLomkn2JKndsKM5c8g8cKnzAiGpGyUUxaR
Supply	200,000,000
Decimal	6
Burn	0%
Proxy	No
Code Visible	Yes
Can Set Fees	Safe
Can take Back Ownership	Safe
Is Mintable	Safe
Can Pause	Yes
Can Blacklist	Safe
Can Burn	Safe

Key Observations and Potential Issues

1. **Outdated Solidity Version**
    - Uses pragma solidity ^0.4.23, which is significantly outdated.
    - Vulnerable to known compiler bugs fixed in later versions.
    - Lacks modern features like SafeMath built-in overflow checks.

2. **Recommendation**: Upgrade to at least 0.8.x for better security and gas optimization.

3. **SafeMath Usage**
    - Implements SafeMath library to prevent arithmetic overflow/underflow.
    - Good practice for this version, but redundant in newer Solidity versions.
    - No issues found in implementation.

4. **Ownership and Centralization Risks**
    - Contract is Ownable with ownership transferred to 0x8dd472b07e8b26e53283e1b8c2ea712258094f2d.
    - Owner has significant control:
        - Can mint new tokens (until finishMinting() is called)

        - Can pause transfers
        - Can freeze tokens
    - CONTINUE_MINTING = false means minting is disabled after initialization, which is good for trust.

5. **Note**: Centralization risk exists if the owner address is compromised.

6. **Initialization**
    - init() function mints 200M tokens to the target address and can only be called once.
    - No re-entrancy risk as it's private and called in constructor.
    - Initialized event emitted properly.

7. **Pausability**
    - paused is set to false by default (via PAUSED constant).
    - Only owner can pause/unpause.
    - Transfer and transferFrom functions respect pause state.
    - No issues found, but users should be aware of this feature.

8. **Minting**
    - mint() and mintAndFreeze() are owner-only and disabled after finishMinting().

- finishMinting() called in init() due to CONTINUE_MINTING = false.
- No unlimited minting possible post-deployment, which is secure.

9. **Freezing Mechanism**
    - Complex implementation allowing tokens to be frozen until a specific timestamp.
    - freezeTo() allows users to freeze their own tokens.
    - mintAndFreeze() allows owner to mint frozen tokens (disabled post-init).
    - releaseOnce() and releaseAll() allow users to release matured frozen tokens.
    - Uses a linked-list-like structure (chains) to track freeze periods.
    - Gas usage is non-deterministic and increases with number of freezes per address.

10. **Potential Issues**:
    - No upper limit on number of freezes, could lead to high gas costs or DoS if abused.
    - toKey() uses assembly with a hardcoded "WISH" mask, which is unconventional but functional.

11. **Recommendation**: Consider adding a maximum freeze count per address.

12. **Burning**
    - burn() allows any user to burn their own tokens.
    - Properly reduces totalSupply_ and emits events.
    - No issues found.

13. **Token Transfers**
    - Standard TRC-20 transfer() and transferFrom() implementations.
    - Protected by whenNotPaused modifier.
    - Uses SafeMath, preventing overflow/underflow.
    - No re-entrancy vulnerabilities observed.

14. **Allowance Race Condition**
    - approve() has the standard ERC-20/TRC-20 race condition warning.
    - Mitigated by increaseApproval() and decreaseApproval() functions.
    - Users should be aware to set allowance to 0 before changing it to a new value.

15. **Gas Optimization**
    - Freezing mechanism could be optimized (e.g., using arrays instead of linked lists).
    - Some functions (e.g., releaseAll()) could consume significant gas with many freezes.

16. **Tron-Specific Considerations**
    - Contract uses TRC-20 interface, compatible with Tron.
    - No Tron-specific opcodes or features (e.g., energy/bandwidth management) explicitly used.
    - Should work on Tron but may need testing for gas/energy costs.

Security Assessment

- **Re-entrancy**: No apparent vulnerabilities.
- **Overflow/Underflow**: Protected by SafeMath.
- **Access Control**: Properly implemented with onlyOwner modifier.
- **Denial of Service**: Possible via excessive freezing, but limited impact.
- **Front-running**: Standard approval race condition exists.

Recommendations

1. **Add Freeze Limits**: Cap the number of concurrent freezes per address.
2. **Documentation**: Add NatSpec comments for public functions.
3. **Testing**: Verify on Tron testnet for gas/energy usage.
4. **Events**: Consider adding more detailed events for freezing operations.

About KREDICT

We believe that people have a fundamental need for security and that the use of secure solutions enables every person to use the Internet and every other connected technology. We aim to provide security consulting services to help others make their solutions more resistant to unauthorized access to data & inadvertent manipulation of the system. We support teams from the design phase through the production to launch and surely after.

The KREDICT team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities & specific attack vectors. The team has reviewed cryptographic protocols and distributed system architecture implementations, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code & networks and build custom tools.

Although we are a small team, we surely believe that we can have a momentous impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please email us at **support@kredict.com**